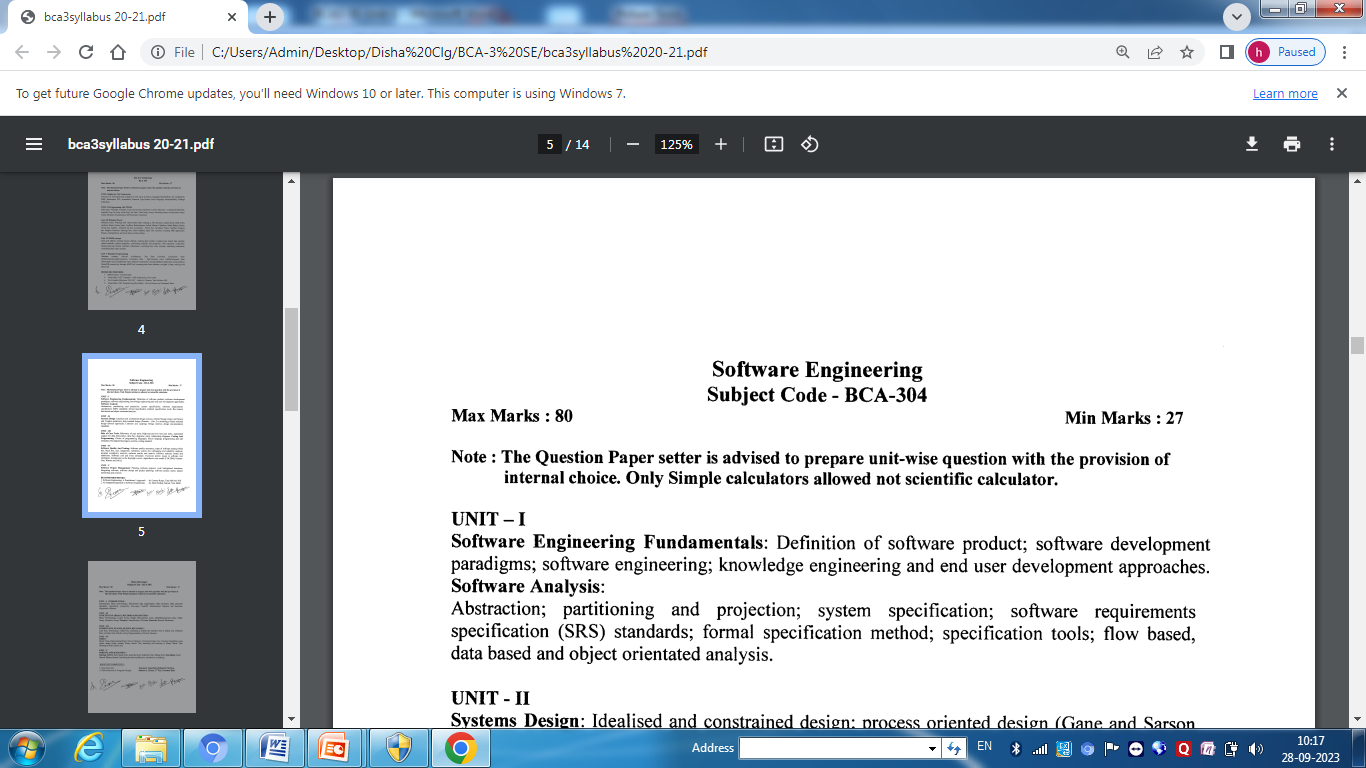
**CLASS BCA-III**

**SUBJECT – Software Engineering**



**Software**

Computer software is the product that software engineers design and build. It encompasses programs that execute within a computer of any size and architecture, documents that encompass hard-copy, and data in form of numbers, text, picture, video, and audio information.

**Program vs. software product**

* Programs are developed by individuals for their personal use. They are therefore, small in size and have limited functionality but software products are extremely large.
* In case of a program, the programmer himself is the sole user but on the other hand, in case of a software product, most users are not involved with the development.
* In case of a program, a single developer is involved but in case of a software product, a large number of developers are involved.
* For a program, the user interface may not be very important, because the programmer is the sole user. On the other hand, for a software product, user interface must be carefully designed and implemented because developers of that product and users of that product are totally different.
* In case of a program, very little documentation is expected, but a software product must be well documented.
* A program can be developed according to the programmer’s individual style of development, but a software product must be developed using the accepted software engineering principles.

**Software Engineering**

* The term software engineering is the product of two words, software, and engineering.
* The software is a collection of integrated programs.
* Engineering is the application of scientific and practical knowledge to invent, design, build, maintain, and improve frameworks, processes, etc.



* **Software Engineering** is an engineering branch related to the evolution of **software product** using well-defined scientific principles, techniques, and procedures. The result of software engineering is an effective and reliable software product.

**According to the Institute of Electrical and Electronics Engineers (IEEE),** software engineering means the application of systematic, disciplined, quantifiable approach to development, operation, and maintenance of software; that is application of the principles of engineering to the software.

The term **'software engineering'** was suggested at conferences organized by NATO in 1968 and 1969 to discuss the 'software crisis'. The software crisis was the name given to the difficulties encountered in developing large, complex systems in the 1960s. It was proposed that the adoption of an engineering approach to software development would reduce the costs of software development and lead to more reliable software.

**Key components of Software Engineering**

Software engineering is the engineering discipline through which software is developed. Commonly the process involves finding out what the client wants, composing this in a list of requirements, designing an architecture capable of supporting all of the requirements, designing, coding, testing and integrating the separate parts, testing the whole, deploying and maintaining the software. Programming is only a small part of software engineering. Following are the key components:-

1. Software Development Life Cycle(SDLC)
2. Software Quality Assurance
3. Software project Management
4. Software Management
5. Computer Aided Software Engineering (CASE)

**Needs of the Software**

Computer software has become a driving force. It is the engine that drives business decision making. It serves as the basis for modern scientific investigation and engineering problem solving. It is a key factor that differentiates modern products and services. It is embedded in systems of all kinds: transportation, medical, telecommunications, military, industrial processes, entertainment, office products, . . . the list is almost endless. Software is virtually inescapable in a modern world. And as we move into the twenty-first century, it will become the driver for new advances in everything from elementary education to genetic engineering.

**Need of software**:

1. The cost of hardware continues to decrease while speed, storage capacity continues to increase. So there is ever increasing need for software.
2. The number of computer users is increasing because cost of hardware is decreasing.
3. The old software systems would not fit into present competitive world the re-engineering of old systems are required with advancement of technology.
4. The demand of software systems for new application area is increasing with the developments in the computer science like artificial intelligence, communication technology, neural network etc.

**Problems/Issues with Software**

Following are the problems with software:

• Why are development costs so high?

• Why does it take so long to get software finished and software slipping out of schedule?

• Why can't we find all the errors before we give the software to customers to make software reliable?

• Why do we continue to have difficulty in measuring progress as software isbeing developed?

• Why do we experience difficulties in maintaining the software?

These and many other questions, related to software development can be solved by adoption of software engineering approaches.It is believed that the only satisfactory solution to the present software crisis can possibly come from a spread of software engineering practices among the engineers, coupled with further advancements to the software engineering discipline itself.

**Software Crisis**

**Software crisis** is a term used in the early days of [computing science](https://en.wikipedia.org/wiki/Computing_science) for the difficulty of writing useful and efficient computer programs in the required time. The software crisis was due to the rapid increases in computer power and the complexity of the problems that could be tackled. With the increase in the complexity of the software, many software problems arose because existing methods were neither sufficient nor up to the mark to solve the software problem.

The crisis is visible itself in several ways:

* Projects running over-budget.
* Projects running over-time.
* [Software was very inefficient.](https://en.wikipedia.org/wiki/Software_optimization)
* Software was of low quality.
* Software often did not meet requirements.
* [Projects were unmanageable and code difficult to maintain.](https://en.wikipedia.org/wiki/Spaghetti_code)
* [Software was never delivered](https://en.wikipedia.org/wiki/Vaporware).

**Software crisis can be classified in two ways:**

[](http://1.bp.blogspot.com/-RCvKZyaClfc/TsiIqkG2nwI/AAAAAAAAAU0/_HN-p2IKHac/s1600/software-problems.jpg)

1.From a programmer’s point of view:-

1. Problem of compatibility.
2. Problem of portability.
3. Problem of documentation.
4. Problem of piracy of software.
5. Problem in coordination to work with other people.
6. Problem that arise during actual run time in the organization.
7. Problem of maintenance in proper manner.

2.From a user’s point of view:-

1. How to choose software from total market availability
2. How to ensure that which software is compatible with our hardware.
3. Problem of viruses.
4. Problems of software bugs.
5. Certain software runs on specific Operating System.
6. Problem of different versions of software.
7. Security problem for protected data in software.

**Software Engineering Problems - (Fundamental problems)**

Software engineering is the systematic approach to the development, operation, maintenance, and retirement of software. There are few fundamental problems that software engineering faces.

**1.The Problem of scale**:    A fundamental problem of software engineering is the problem of scale; development of a very large system requires a very different set of methods compared to developing a small system. Any large project involves the use of technology and project management.

For software projects, by technology we mean the methods, procedures, and tools that are used. In small projects, informal methods for development and management can be used. However, for large projects, both have to be much more formal.

**2.Cost, schedule and quality**: The cost of developing a system is the cost of the resources used for the system, which, in the case of software, are the manpower, hardware, software, and the other support resources. Generally, the manpower component is predominant, as software development is largely labor-intensive and the cost of the computing systems is now quite low.

Hence, the cost of software project is measured in terms of person-months, i.e. the cost is considered to be the total number of person-months spent in the project. Schedule is an important factor in many projects. Business trends are dictating that the time to market of a product should be reduced; that is, the cycle time from concept to delivery should be small.

One of the major factors driving any production discipline is quality. We can view **quality of a software** product as having three dimensions:

**Product Operation**: Correctness (performs intended purpose correctly), reliability (perform required function under given condition), efficiency (use resource in efficient manner), usability (easy to use) and integrity (controlled data and software modification), robustness (continue to operate with invalid input).  
**Product Transition**: Portability(transfer from one platform to other) and reusability (reuse of existing code).  
**Product Revisio**n: Flexibility (can support changing needs), testability (different testing method can be applied) and modifiability or evolvability.

**3.The Problem of consistency**: Though high quality, low cost and small cycle time are the primary objectives of any project, for an organization there is another goal: consistency. An organization involved in software development does not just want low cost and high quality for a project, but it wants these consistently for all projects.

Businesses also employ software engineers to create customized software and address vulnerabilities before they happen. This makes sense when we think of the complexity of the tasks that the average professional carries out, tasks like holding meetings in real time with collaborators oceans away. Even when engineering principles aren’t necessary for safety, sound design can increase efficiency and decrease costs.

You’ll find a diverse group of employers advertising for true software engineers. Disney Interactive Media is among the companies seeking software developers who are familiar with the software development life cycle.

**Characteristics of Software**

It is important to examine the characteristics of software that make it different from other things that human beings build. When hardware is built, the human creative process (analysis, design, construction, testing) is ultimately translated into a physical form.

•Software is a logical rather than a physical system element. Therefore, software has characteristics that are considerably different than those of hardware.

•Software is developed or engineered; it is not manufactured in the classical sense.

•Software doesn't "wear out."

•Although the industry is moving toward component-based assembly, most software continues to be custom built.

**Quality of Software Product**

|  |  |
| --- | --- |
| **Attribute** | **Definition** |
| **Correctness** | The extent to which a program satisfies its specification. |
| **Reliability** | The property that defines how well the software meets its requirements. |
| **Efficiency** | A factor in all issues relating to the execution of software; it includes suchconsideration as response time, memory requirements and throughput. |
| **Usability** | The effort required to learn and operate the software properly, is an importantproperty that emphasizes the human aspect of the system. |
| **Portability** | The effort required to transfer the software from one hardware configuration toanother. |
| **Reusability** | The extent to which parts of the software can be reused in other related applications. |
| **Interoperability** | The effort required to couple the system with other systems. |
| **Maintainability** | The effort required to locate and fix errors in operating programs. |
| **Flexibility** | The effort required to modify an operational program (perhaps to enhance its functionality). |
| **Testability** | The effort required to test to ensure that the systems or a module performs its intended function. |

**Software Engineering Approach**

Software development life cycle describe phases of the software development and the order in which those phases are executed. Each phase produces deliverables required by the next phase in the SDLC.

**There are following six phases in every Software development life cycle model:**

1. Preliminary Investigation
2. Requirement gathering and analysis
3. Design
4. Implementation or coding
5. Testing and Deployment
6. Maintenance

**1) Preliminary Investigation:**  this phase commences with discussion on the requests made by the user. The requests can be for a new system or modifying the existing system.

**Technical Feasibility**: An estimate is made of whether the identified user needs can be satisfied with the current hardware and software technologies or not.

**Economic Feasibility**: are there sufficient benefits in creating the system to make the costs acceptable? Whether the investment needed to implement the system will be recovered at later stages or not.

**Operational Feasibility**: will the system be used if it is developed and implemented? Will there be resistance from users that will undermine the possible application benefit?

Feasibility report is the outcome of this phase. Feasibility study report consists of:

(i) a preamble that sets stage for the project (ii) objective of the project (iii) study of existing system (iv)proposed system (v) and Technical, economical and operational feasibility study.

**Preliminary report decides whether to go for project or not.**

**2) Requirement gathering and analysis:**  Business requirements are gathered in this phase. This phase is the main focus of the project managers and stake holders. Meetings with managers, stake holders and users are held in order to determine the requirements like; who is going to use the system? How will they use the system?  What data should be input into the system?  What data should be output by the system?  These are general questions that get answered during a requirements gathering phase. After requirement gathering these requirements are analyzed for their validity and the possibility of incorporating the requirements in the system to be development is also studied.

Finally, a Requirement Specification document is created which serves the purpose of guideline for the next phase of the model.

**3)  Design:**  In this phase the system and software design is prepared from the requirement specifications which were studied in the first phase. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model.

**4)  Implementation / Coding:**  On receiving system design documents, the work is divided in modules/units and actual coding is started. Since, in this phase the code is produced so it is the main focus for the developer. This is the longest phase of the software development life cycle.

**5)** [**Testing**](http://istqbexamcertification.com/what-is-a-software-testing/) **and Deployment:**  After the code is developed it is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. During this phase unit testing, integration testing, system testing, acceptance testing are done. After successful testing the product is delivered / deployed to the customer for their use.

**6) Maintenance:** Once when the customers starts using the developed system then the actual problems comes up and needs to be solved from time to time. This process where the care is taken for the developed product is known as maintenance.

**Software Engineering Principles**

**Rigor and Formality:** software development is creative activity. There is inherent tendency in any creative process to be neither precise nor accurate. It only through rigorous approach that we can produce more reliable products, controls their costs, and increases our confidence in product reliability. The highest degree of rigor is formality. Formality is stronger requirement than rigor.

There is no need to be always formal during design, but the software engineer must know when to be formal, and when to be rigorous. In case of critical parts formal description of intended functions and a formal approach to their assessment must be applied using logical statement of a language. Well understood and standard parts would require rigor using natural language.

**Rigor and formality improves reliability and verifiability of software product. It also has beneficial effect on maintainability, reusability, portability, understandability and interoperability. For e.g. rigorous and formal documentation can increase all above mentioned qualities.**

### Separation of Concerns: Separation of concerns allows us to deal with different individual aspects of a problem, so that we can concentrate on each separately. Separation of concerns is a commonsense practice that we try to follow in our everyday life to tackle difficulties we encounter.

The only way to master the complexity of the project is to separate the different concerns first of all, one should try to isolate issues that are less intimately related to the others.

Some of the concerns pertaining to features of the product are: functions to offer, expected reliability, space and time efficiency, user interface etc.

Some of the concerns pertaining to development process are: team organization and structure, scheduling, error recovery procedure, design strategy etc.

Other concerns are: economic and financial matters.

In view of this, it makes sense to separate handling of different concerns. This can be done by dealing with different concerns at different times in the software life cycle or assigning of different concerns to different skilled people. E.g.one can work for correctness of a function first, and then can work for improvement of efficiency of the function.

### Modularity

The principle of modularity is a specialization of the principle of separation of concerns. Following the principle of modularity implies separating software into components according to functionality and responsibility. It also has beneficial effect on maintainability, reusability, testing debugging and understandability and supports component based software development.

### Abstraction

The principle of abstraction is special case of the principle of separation of concerns. Following the principle of abstraction implies separating the behavior of software components from their implementation. It requires learning to look at software and software components from two points of view: what it does, and how it does it.

Abstraction is powerful technique practiced by engineers of all fields for mastering complexity.

### Anticipation of Change

Changes are introduced in software due to: elimination of error that were not detected before release of software, need for supporting evolution of the software as new requirements arise or old requirements change. This is why maintainability is major software quality. Reusability is another quality that is strongly affected by anticipation of change. Anticipation of change is a principle that we can use to achieve evolvability.

If the problem to be solved is complex then it is not reasonable to assume that the best solution will be worked out in a short period of time. The clients do, however, want a timely solution. In most cases, they are not willing to wait until the perfect solution is worked out. They want a reasonable solution soon; perfection can come later. To develop a timely solution, software developers need to know the requirements: how the software should behave. Preliminary requirements need to be worked out early, but it should be possible to make changes in the requirements as learning progresses.

Coupling is a major obstacle to change. If two components are strongly coupled then it is likely that changing one will not work without changing the other.

Cohesiveness has a positive effect on ease of change. Cohesive components are easier to reuse when requirements change. If a component has several tasks rolled up into one package, it is likely that it will need to be split up when changes are made.

### Generality

The principle of generality is closely related to the principle of anticipation of change. It is important in designing software that is free from unnatural restrictions and limitations

Every time it is required to solve a problem, try to focus on the discovery of a more general problem that may be hidden behind the problem at hand. It may happen that the generalized problem is not more complex- it may even be simpler-than the original problem. Being more general, it likely that the solution to the generalized problem has more potential for being reused. It may even happen that the solution is already proved by some existing software package.

### Incremental Development

In this process, you build the software in small increments; an incremental software development process simplifies verification. If you develop software by adding small increments of functionality then, for verification, you only need to deal with the added portion.

One way of applying the incrementality principle consists of identifying useful early subsets ofsoftware that may be developed and delivered to customer, in order to get early feedback. This allows the application to evolve in a controlled manner in cases where the initial requirements are not stable or fully understood.

Initial version may focus on correctness, usability and reliability after that on performance and efficiency- it can be done incrementality.

### Consistency

The principle of consistency is recognition of the fact that it is easier to do things in a familiar context. For example, coding style is a consistent manner of laying out code text. First it makes reading the code easier. Second, it allows programmers to automate part of the skills required in code entry, freeing the programmer's mind to deal with more important issues.

Consistency serves two purposes in designing graphical user interfaces. First, a consistent look and feel makes it easier for users to learn to use software. Once the basic elements of dealing with an interface are learned, they do not have to be relearned for a different software application. Second, a consistent user interface promotes reuse of the interface components.

Principle of consistency applies to object-oriented class libraries. As the available libraries grow more and more complex it is essential that they be designed to present a consistent interface to the client. For example, most data collection structures support adding new data items. It is much easier to learn to use the collections if the name “add” is always used for this kind of operation.

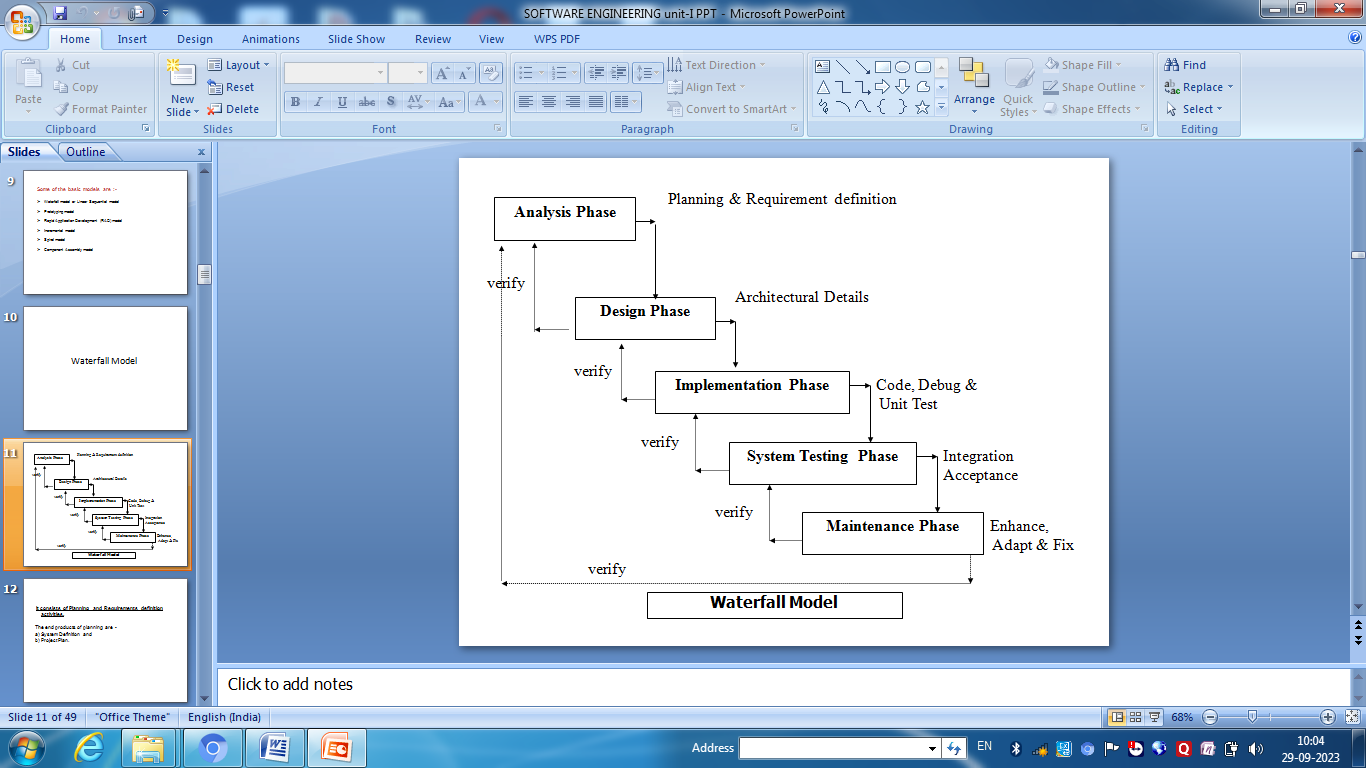
### Software development Model -

* 1. **Waterfall model**
  2. **Prototype Model**
  3. **Spiral Model**
  4. **Iterative Model**
  5. **Incremental Model**

### 1.Waterfall model-

The Waterfall Model was first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**.  It is very simple to understand and use.  In a waterfall model, each phase must be completed fully before the next phase can begin. This type of model is basically used for the project which is small and there are no uncertain requirements. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. In this model the testing starts only after the development is complete. In **waterfall model phases** do not overlap.

**Diagram of Waterfall-model:**



**Advantages of waterfall model:**

* This model is simple and easy to understand and use.
* It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
* In this model phases are processed and completed one at a time. Phases do not overlap.
* Waterfall model works well for smaller projects where requirements are very well understood.

**Disadvantages of waterfall model:**

* Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
* No working software is produced until late during the life cycle.
* High amounts of risk and uncertainty.
* Not a good model for complex and object-oriented projects.
* Poor model for long and ongoing projects.
* Not suitable for the projects where requirements are at a moderate to high risk of changing.

**When to use the waterfall model:**

* This model is used only when the requirements are very well known, clear and fixed.
* Product definition is stable.
* Technology is understood.
* There are no ambiguous requirements
* Ample resources with required expertise are available freely
* The project is short.

Very less customer interaction is involved during the development of the product. Once the product is ready then only it can be demoed to the end users. Once the product is developed and if any failure occurs then the cost of fixing such issues are very high, because we need to update everywhere from document till the logic.

**2.Spiral model-**

The spiral model is similar to the [incremental model](http://istqbexamcertification.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/), with more emphasis placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spiral, starting in the planning phase, requirements is gathered and risk is assessed. Each subsequent spiral builds on the baseline spiral.

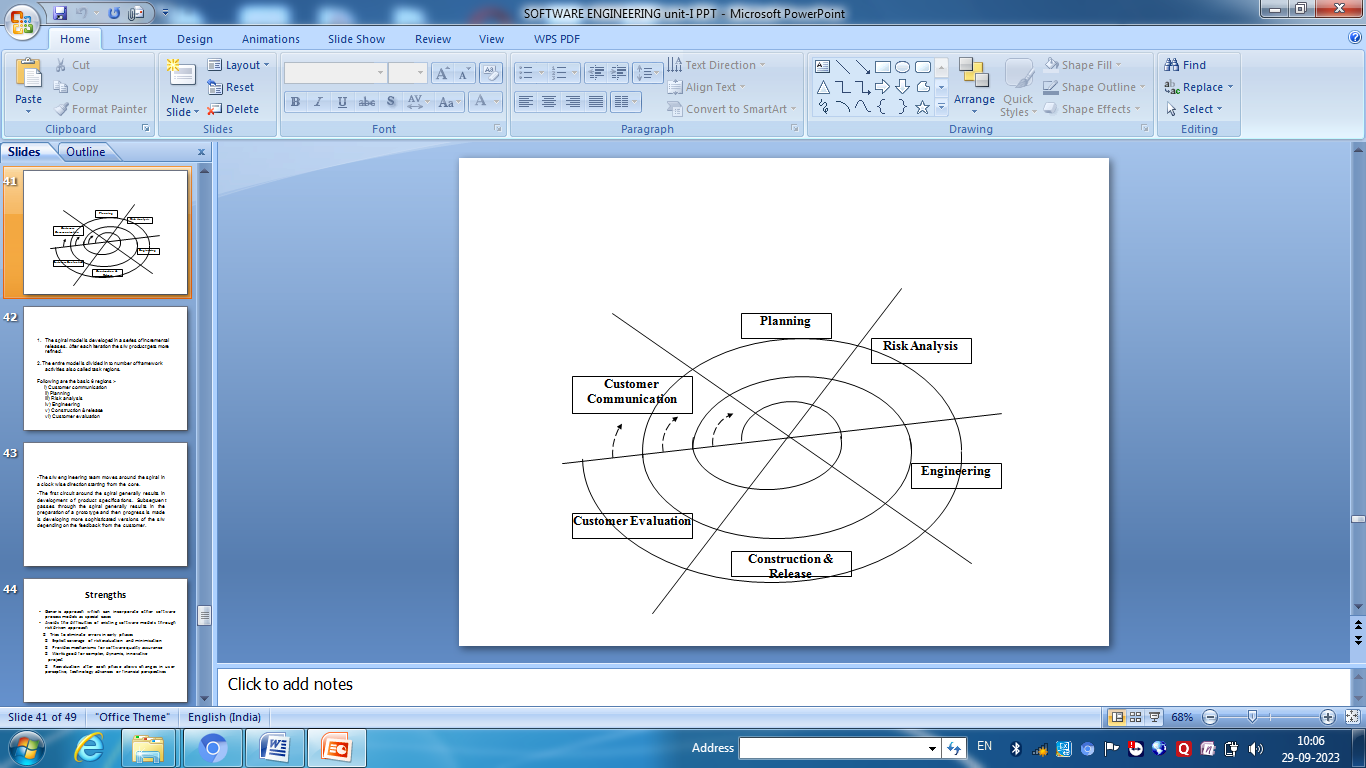
**Planning Phase: In planning phase r**equirements are gathered.Requirements like ‘BRS’ that is ‘Business Requirement Specifications’ and ‘SRS’ that is ‘System Requirement specifications’.

**Risk Analysis:** In the **risk analysis phase**, a process is undertaken to identify risk and alternate solutions.  A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

**Engineering Phase:** In this phase software is **developed**, along with [testing](http://istqbexamcertification.com/what-is-a-software-testing/) at the end of the phase. Hence in this phase the development and testing is done.

E**valuation phase:** This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

**Diagram of Spiral model:**



**Advantages of Spiral model:**

* High amount of risk analysis hence, avoidance of Risk is enhanced.
* Good for large, complex, dynamic, and mission-critical projects.
* Specifies mechanism for software quality assurance activities.
* Strong approval and documentation control.
* Additional Functionality can be added at a later date.
* Software is produced early in the [software life cycle](http://istqbexamcertification.com/what-are-the-software-development-life-cycle-phases/).
* Estimation of budget and schedule gets realistic as the work progresses.

**Disadvantages of Spiral model:**

* Can be a costly model to use.
* It is difficult to estimate budget and schedule in the beginning as some of the analysis is not done until the design of the software is created.
* Risk analysis requires highly specific expertise.
* Project’s success is highly dependent on the risk analysis phase.
* Doesn’t work well for smaller projects.

**When to use Spiral model:**

* When costs and risk evaluation is important
* For medium to high-risk projects
* Users are unsure of their needs
* Requirements are complex
* New product line
* Significant changes are expected (research and exploration)

# 3.Prototype model

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. 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. The prototypes are usually not complete systems and many of the details are not built in the prototype. The goal is to provide a system with overall functionality.

**Diagram of Prototype model:**

[](http://istqbexamcertification.com/wp-content/uploads/2012/01/Prototype-model.jpg)

**Advantages of Prototype model:**

* Users are actively involved in the development
* Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.
* Errors can be detected much earlier.
* Quicker user feedback is available leading to better solutions.
* Missing functionality can be identified easily
* Confusing or difficult functions can be identified.
* Requirements validation is quick.
* Quick implementation of, incomplete, but functional application is possible.
* Risk associated with project is reduced.

**Disadvantages of Prototype model:**

* Leads to implementing and then repairing way of building systems.
* Developer loses focus of the real purpose of prototype and compromise with the quality of the product. For example they may apply some of the inefficient algorithms or inappropriate programming languages for developing the prototype.
* Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
* Prototyping can lead to false expectations. It often creates a situation where user believes that the development of the system is finished when it is not.

**When to use Prototype model:**

* Prototype model should be used when the desired system needs to have a lot of interaction with the end users.
* Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.
* Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

# 4.Iterative model

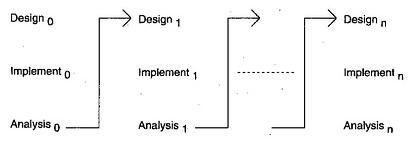
An iterative [life cycle model](http://istqbexamcertification.com/what-are-the-software-development-models/) does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software for each cycle of the model.

For example:

[](http://istqbexamcertification.com/wp-content/uploads/2012/01/Iterative-model-example.jpg)

In the diagram above when we work **iteratively** we create rough product or product piece inone iteration, then review it and improve it in next iteration and so on until it’s finished. As shown in the image above, in the first iteration the whole painting is sketched roughly, then in the second iteration colors are filled and in the third iteration finishing is done. Hence, in iterative model the whole product is developed step by step.

**Diagram of Iterative model:**

[](http://istqbexamcertification.com/wp-content/uploads/2012/01/Iterative-model.jpg)

**Advantages of Iterative model:**

* In iterative model we can only create a high-level design of the application before we actually begin to build the product and define the design solution for the entire product. Later on we can design and built a skeleton version of that, and then evolved the design based on what had been built.
* In iterative model we are building and improving the product step by step. Hence we can track the defects at early stages. This avoids the downward flow of the defects.
* In iterative model we can get the reliable user feedback. When presenting sketches and blueprints of the product to users for their feedback, we are effectively asking them to imagine how the product will work.
* In iterative model less time is spent on documenting and more time is given for designing.

**Disadvantages of Iterative model:**

* Each phase of an iteration is rigid with no overlaps
* Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle

**When to use iterative model:**

* Requirements of the complete system are clearly defined and understood.
* When the project is big.
* Major requirements must be defined; however, some details can evolve with time.

**5.Increment Model**

In incremental model the whole requirement is divided into various builds. Multiple development cycles take place here, making the life cycle a [“multi-waterfall” cycle](http://istqbexamcertification.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/).  Cycles are divided up into smaller, more easily managed modules.  Each module passes through the requirements, design, implementation and [testing](http://istqbexamcertification.com/what-is-a-software-testing/) phases. A working version of software is produced during the first module, so you have working software early on during the [software life cycle](http://istqbexamcertification.com/what-are-the-software-development-life-cycle-phases/). Each subsequent release of the module adds function to the previous release. The process continues till the complete system is achieved.

For example:

[](http://istqbexamcertification.com/wp-content/uploads/2012/01/Incremental-model_11.jpg)

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

**Diagram of Incremental model:**

[](http://istqbexamcertification.com/wp-content/uploads/2012/01/Incremental_model.jpg)

**Advantages of Incremental model:**

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

**Disadvantages of Incremental model:**

* Needs good planning and design.
* Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
* Total cost is higher than [waterfall](http://istqbexamcertification.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/).

**When to use the Incremental model:**

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

Software Analysis

## Abstraction

An abstraction is a tool that enables a designer to consider a component at an abstract level without bothering about the internal details of the implementation. Abstraction can be used for existing element as well as the component being designed.

Here, there are two common abstraction mechanisms

1. Functional Abstraction
2. Data Abstraction

### Functional Abstraction

1. A module is specified by the method it performs.
2. The details of the algorithm to accomplish the functions are not visible to the user of the function.

Functional abstraction forms the basis for **Function oriented design approaches**.

### Data Abstraction

Details of the data elements are not visible to the users of data. Data Abstraction forms the basis for **Object Oriented design approaches**.

**Partitioning**

For small problem, we can handle the entire problem at once but for the significant problem, divide the problems and conquer the problem it means to divide the problem into smaller pieces so that each piece can be captured separately.

For software design, the goal is to divide the problem into manageable pieces.

### Benefits of Problem Partitioning

1. Software is easy to understand
2. Software becomes simple
3. Software is easy to test
4. Software is easy to modify
5. Software is easy to maintain
6. Software is easy to expand

These pieces cannot be entirely independent of each other as they together form the system. They have to cooperate and communicate to solve the problem. This communication adds complexity.

# Software Requirement Specifications

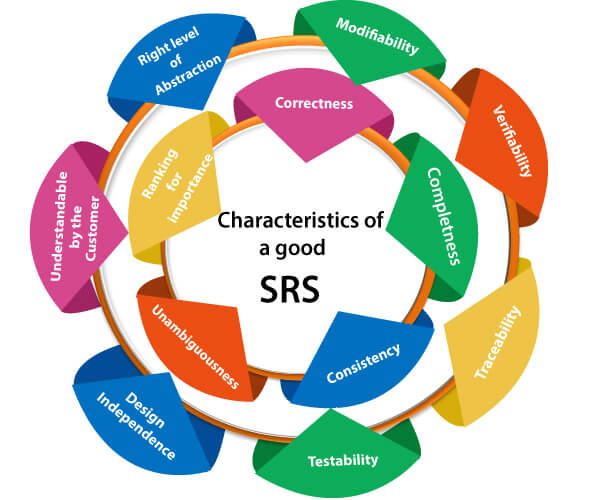
The production of the requirements stage of the software development process is **Software Requirements Specifications (SRS)** (also called a **requirements document**). This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited and analyzed.

**SRS** is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements.

The SRS is a specification for a specific software product, program, or set of applications that perform particular functions in a specific environment. It serves several goals depending on who is writing it. First, the SRS could be written by the client of a system. Second, the SRS could be written by a developer of the system.

The two methods create entirely various situations and establish different purposes for the document altogether. **The first case**, SRS, is used to define the needs and expectation of the users. **The second case**, SRS, is written for various purposes and serves as a contract document between customer and developer.

## Characteristics of good SRS



**Following are the features of a good SRS document:**

**1. Correctness:** User review is used to provide the accuracy of requirements stated in the SRS. SRS is said to be perfect if it covers all the needs that are truly expected from the system.

**2. Completeness:** The SRS is complete if, and only if, it includes the following elements:

**(1).** All essential requirements, whether relating to functionality, performance, design, constraints, attributes, or external interfaces.

**(2).** Definition of their responses of the software to all realizable classes of input data in all available categories of situations.

**(3).** Full labels and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.

**3. Consistency:** The SRS is consistent if, and only if, no subset of individual requirements described in its conflict. There are three types of possible conflict in the SRS:

**(1).** The specified characteristics of real-world objects may conflicts. For example,

(a) The format of an output report may be described in one requirement as tabular but in another as textual.

(b) One condition may state that all lights shall be green while another states that all lights shall be blue.

**(2).** There may be a reasonable or temporal conflict between the two specified actions. For example,

(a) One requirement may determine that the program will add two inputs, and another may determine that the program will multiply them.

(b) One condition may state that "A" must always follow "B," while other requires that "A and B" co-occurs.

**(3).** Two or more requirements may define the same real-world object but use different terms for that object. For example, a program's request for user input may be called a "prompt" in one requirement's and a "cue" in another. The use of standard terminology and descriptions promotes consistency.

**4. Unambiguousness:** SRS is unambiguous when every fixed requirement has only one interpretation. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.

**5. Ranking for importance and stability:** The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.

Typically, all requirements are not equally important. Some prerequisites may be essential, especially for life-critical applications, while others may be desirable. Each element should be identified to make these differences clear and explicit. Another way to rank requirements is to distinguish classes of items as essential, conditional, and optional.

**6. Modifiability:** SRS should be made as modifiable as likely and should be capable of quickly obtain changes to the system to some extent. Modifications should be perfectly indexed and cross-referenced.

**7. Verifiability:** SRS is correct when the specified requirements can be verified with a cost-effective system to check whether the final software meets those requirements. The requirements are verified with the help of reviews.

**8. Traceability:** The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancement documentation.

**There are two types of Traceability:**

**1. Backward Traceability:** This depends upon each requirement explicitly referencing its source in earlier documents.

**2. Forward Traceability:** This depends upon each element in the SRS having a unique name or reference number.

The forward traceability of the SRS is especially crucial when the software product enters the operation and maintenance phase. As code and design document is modified, it is necessary to be able to ascertain the complete set of requirements that may be concerned by those modifications.

**9. Design Independence:** There should be an option to select from multiple design alternatives for the final system. More specifically, the SRS should not contain any implementation details.

**10. Testability:** An SRS should be written in such a method that it is simple to generate test cases and test plans from the report.

**11. Understandable by the customer:** An end user may be an expert in his/her explicit domain but might not be trained in computer science. Hence, the purpose of formal notations and symbols should be avoided too as much extent as possible. The language should be kept simple and clear.

**12. The right level of abstraction:** If the SRS is written for the requirements stage, the details should be explained explicitly. Whereas,for a feasibility study, fewer analysis can be used. Hence, the level of abstraction modifies according to the objective of the SRS.

## Properties of a good SRS document

**The essential properties of a good SRS document are the following:**

**Concise:** The SRS report should be concise and at the same time, unambiguous, consistent, and complete. Verbose and irrelevant descriptions decrease readability and also increase error possibilities.

**Structured:** It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the user requirements. Often, user requirements evolve over a period of time. Therefore, to make the modifications to the SRS document easy, it is vital to make the report well-structured.

**Black-box view:** It should only define what the system should do and refrain from stating how to do these. This means that the SRS document should define the external behavior of the system and not discuss the implementation issues. The SRS report should view the system to be developed as a black box and should define the externally visible behavior of the system. For this reason, the SRS report is also known as the black-box specification of a system.

**Conceptual integrity:** It should show conceptual integrity so that the reader can merely understand it. Response to undesired events: It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.

**Verifiable:** All requirements of the system, as documented in the SRS document, should be correct. This means that it should be possible to decide whether or not requirements have been met in an implementation.

**Assingment Question**

1. Explain software and characteristics of software.
2. [What are Various Software Engineering Problems (Fundamental problems)? Explain](http://ecomputernotes.com/software-engineering/what-are-various-software-engineering-problems-explain)
3. What are different phases of Software Engineering?
4. Explain various phases of software development life cycle?
5. What is Software Engineering Approach?
6. What is Software Engineering Paradigm?
7. What is quality of Software Product?
8. What is Waterfall model? What are advantages, disadvantages and when to use it?
9. What is Prototype model- advantages, disadvantages and when to use it?
10. .Explain Software Engineering Principles .
11. .What is Spiral model- advantages, disadvantages and when to use it?
12. Difference between prototype (evolutionary) and spiral model .
13. Difference between prototype and water fall model .