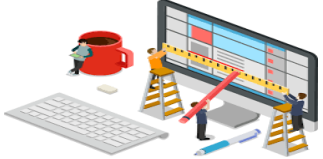
Software Engineering

Summarized & Presented By Dr.Engineer

IBRAHIM ESKANDAR

Outline of the Course

1 Introduction
2 Software processes
3 Agile software development
4 Requirements engineering
5 System modeling
6 Architectural design
7 Design and implementation
8 Software testing
9 Software evolution

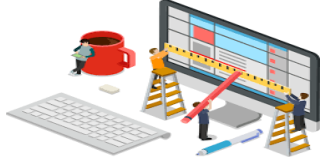


1. **Understand what software engineering is, why it is hard and why it is important.**

2. **Understand that the development of different types of software system may require different software engineering techniques.**

3. **Understand ethical and professional issues that are important for software engineers.**

4. **Knowledge about systems around.**



Software engineering why it is important?

Software engineering is essential for the functioning of government, society, and national and international businesses and institutions. We can't run the modern world without software. National infrastructures and utilities are controlled by computer-based systems, and most electrical products include a computer and controlling software.

Industrial manufacturing and distribution is completely computerized, as is the financial system. Entertainment, including the music industry, computer games, and film and television, is software-intensive. More than 75% of the world's populations have a software-controlled mobile phone, and, by 2016, almost all of these will be Internet-enabled.

Software systems become extremely complex, difficult to understand, and expensive to change WHY it's hard?

Software systems are abstract and intangible. They are not constrained by the properties of materials, nor are they governed by physical laws or by manufacturing processes.

This simplifies software engineering, as there are no natural limits to the potential of software. However, because of the lack of physical constraints, software systems can quickly become extremely complex, difficult to understand, and expensive to change.

Summarized & Presented By Dr.Engineer.IBRAHIM ESKANDAR

Software \ Engineering

Engineering: application of tools methods and science to find cost effective solution.

Software Engineering: it is defined as systematic disciplined and quantifiable approach to development – operation and maintenance of software.

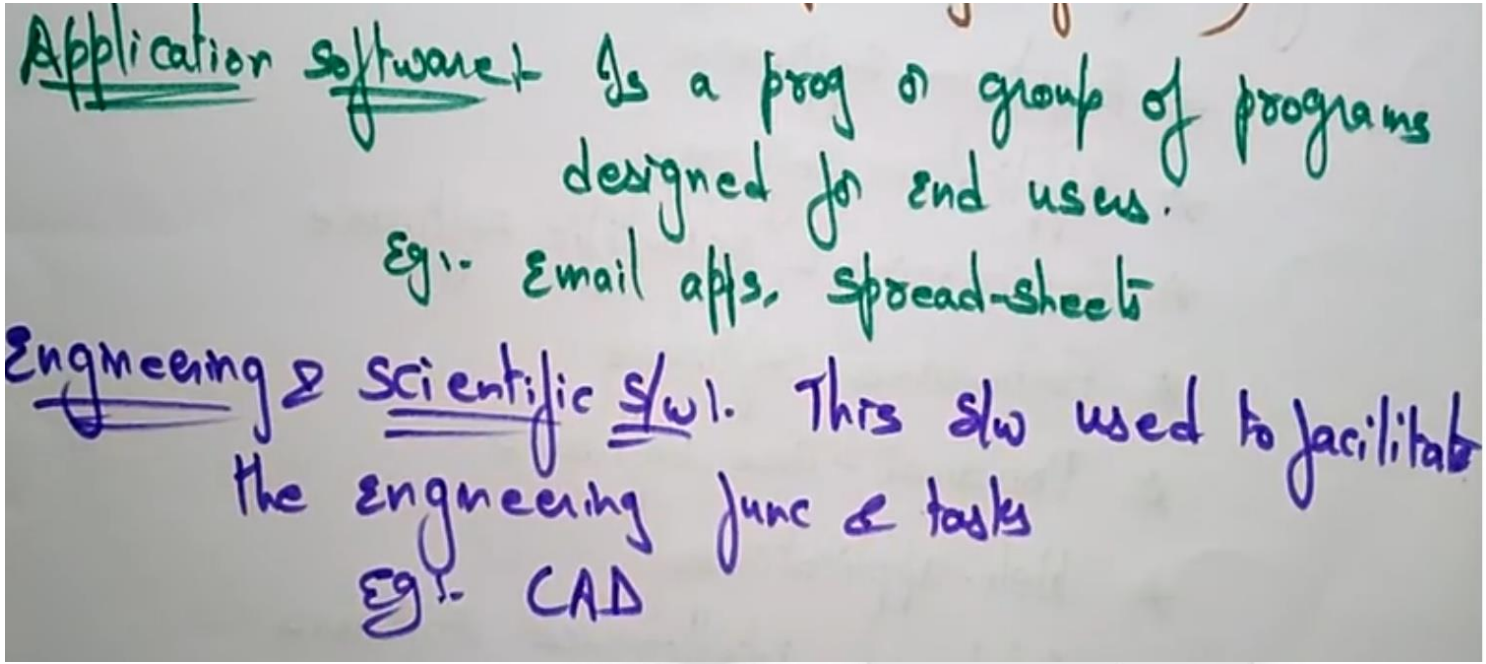
Characteristics of Software: developed or engineered not manufactured – customized – not wave-out.

Software Quality: a quality product do exactly what the users want it to do.

Software Quality Factors: portability - usability – reusability – correctness – maintainability

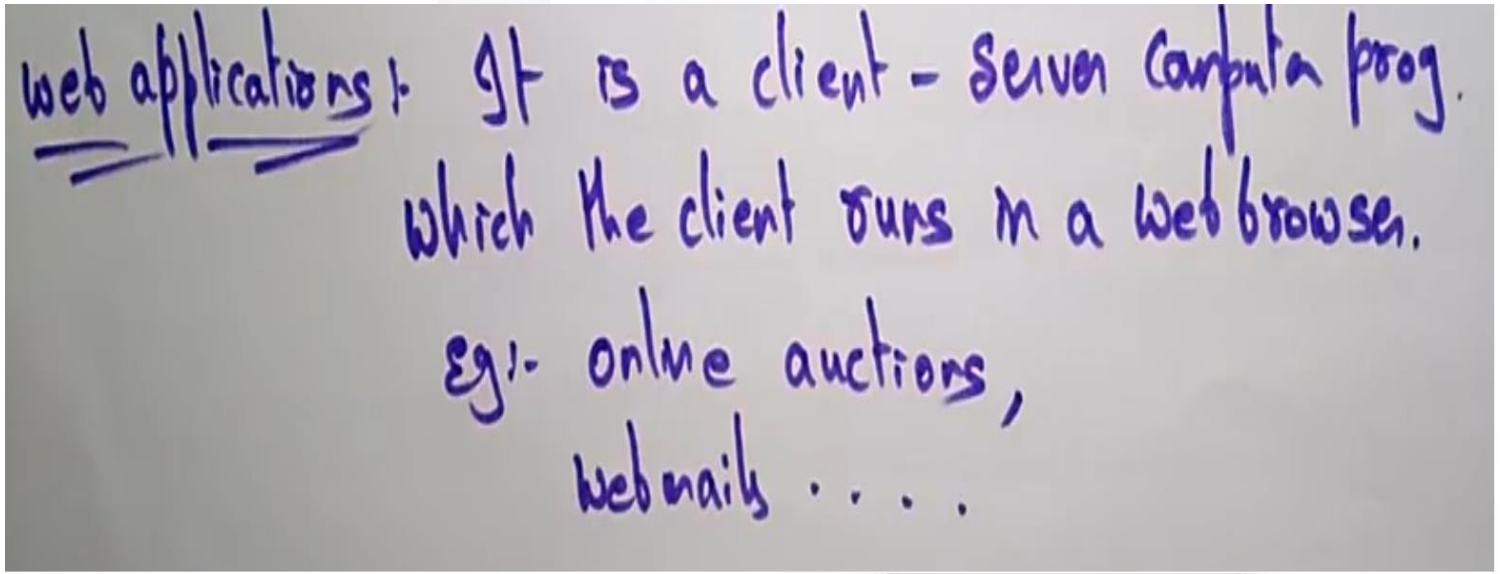
- The Seven Broad Categories of Software are challenges for Software engineers.
  - ✯ System Software
  - ✯ Application Software
  - ✯ Engineering & scientific Software
  - ✯ Embedded Software
  - ✯ Product-line Software
  - ✯ Web-applications
  - ✯ Artificial intelligence Software

1. System software: It is a collection of program written to service other program
   Eg: OS (operating system)
2. Application software: Is a prog or group of programs designed for end users.
   Eg: Email apps, Spread-sheets



4. Embedded S/W :- It resides in read-only memory
- It is used to control products & sys for consumer & industrial markets

Eg:. GPS devices, factory robots, calculators, modern smart-watches

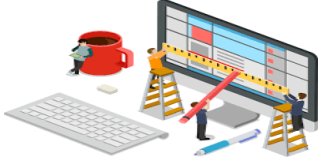


Artificial intelligence s/w :-

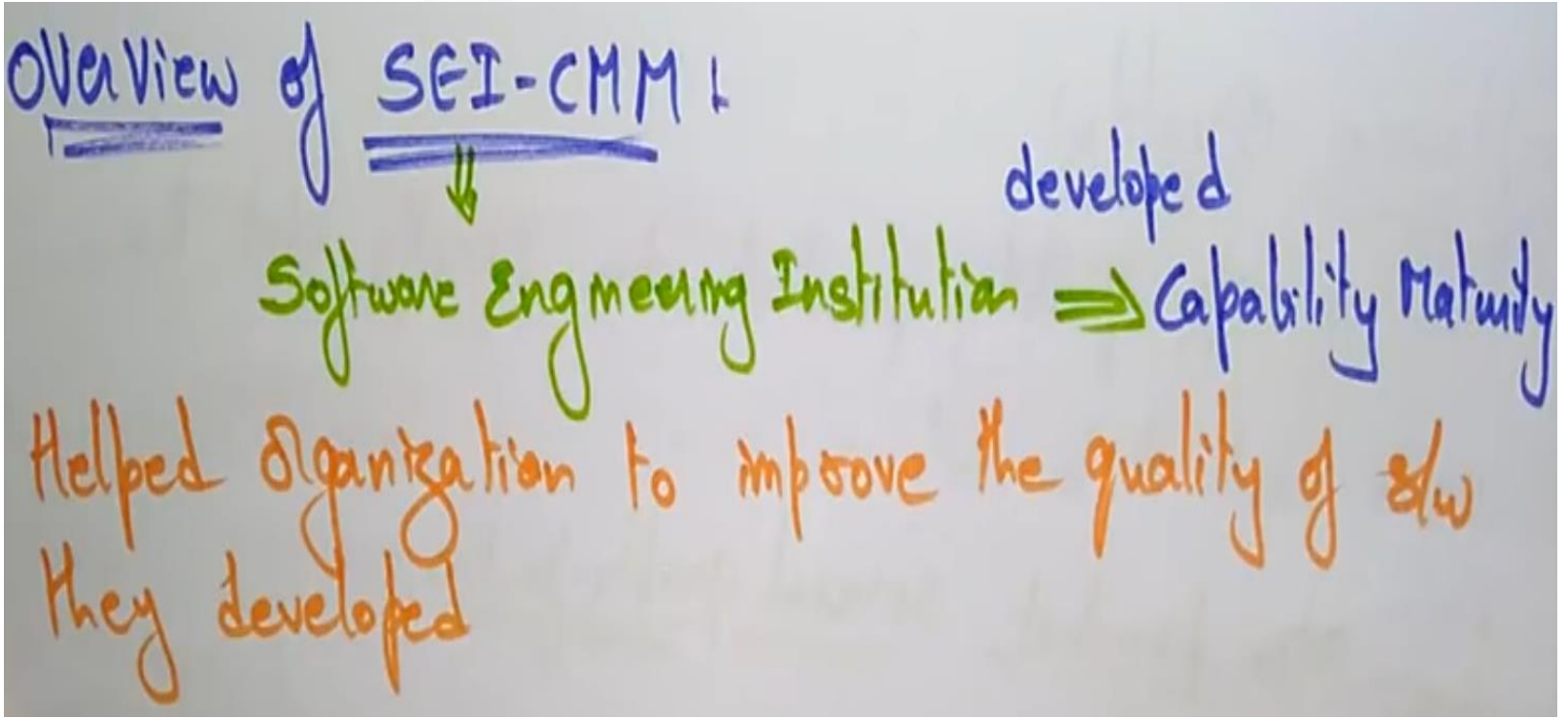
m/c learning includes alg that are developed to tell a computa how to respond to something by example.

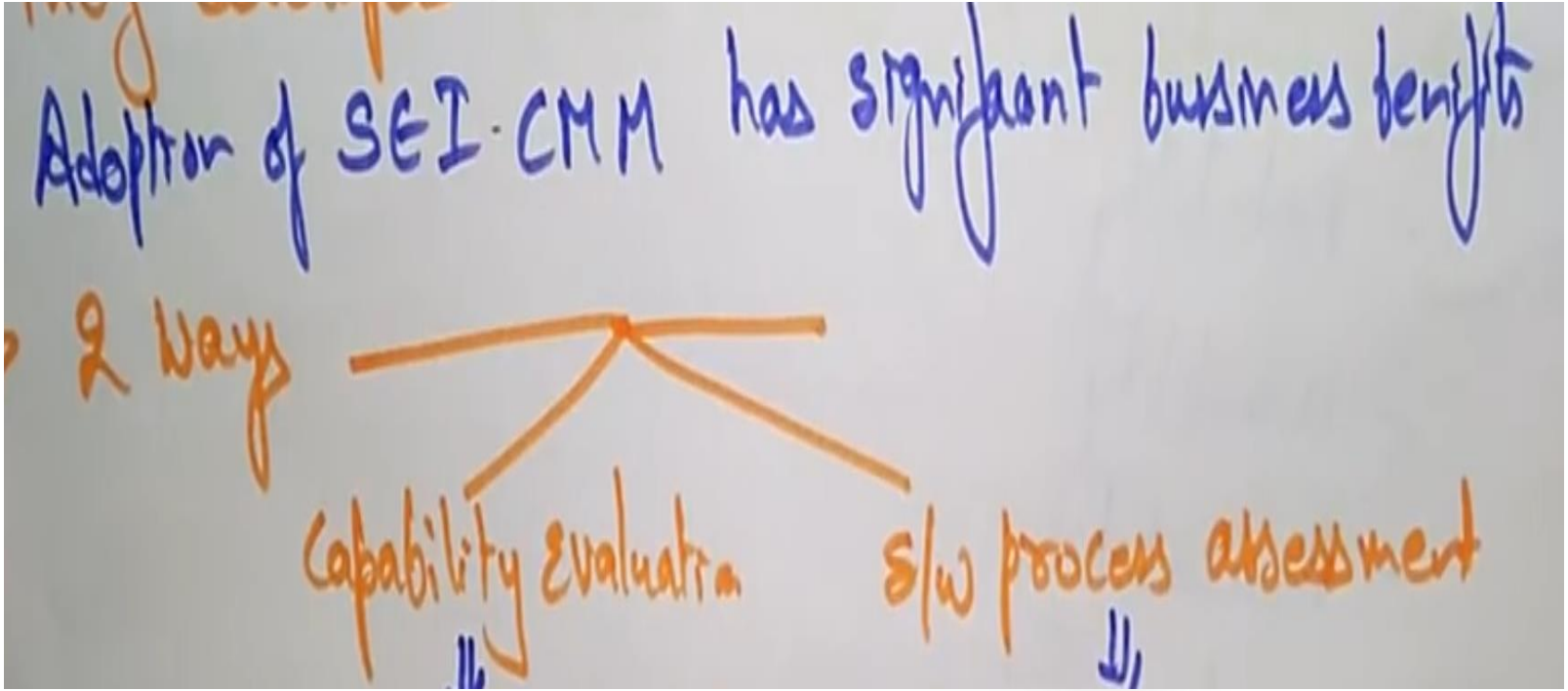
- It uses a structure as close as possible to human brain.

Eg:- Speech recognition



Software Engineering Institute – Capability Maturity Model





Helped Organization to improve the quality of s/w they developed

* Adoption of SEI-CMM has significant business benefits

* 2 ways

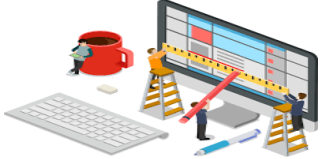
- Capability Evaluation ⇓ provides a way to access the s/w process capability
- s/w process assessment ⇓ It is used by an organization with the objective to improve the process capability.

05:13

⇒ SEI CMM classifies s/w development industries into 5 maturity levels.

Level 1: Initial

A s/w development organization at this level is characterized by ad hoc activities.



Level 1: Initial

A s/w development organization at this level is characterized by ad hoc activities.

level 2: Repeatable

At this level, the basic project management practi such as tracking cost & schedule are established

level 3: defined.

At this level processes for both management & development activities are defined & documented.

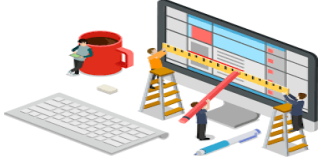
level 4: Managed

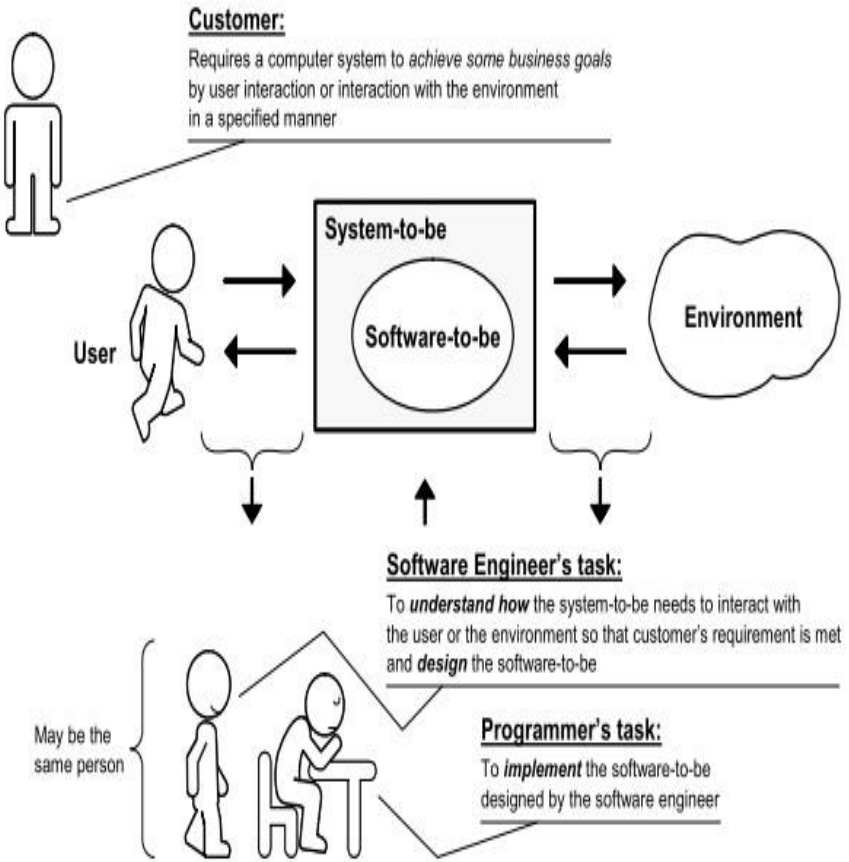
At this level, the focus is on s/w metrics.

level 5: optimizing

At this stage, processes & product metrics are collected.

Ch1





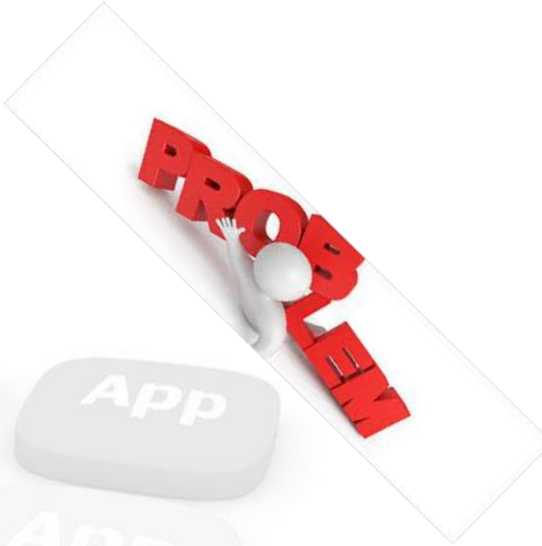
Software engineering is essential for the functioning of government, society and international businesses and organizations. Modern word is developed and run smoothly by blessing of Allah firstly then by software engineering.

-Abstract and intangible.

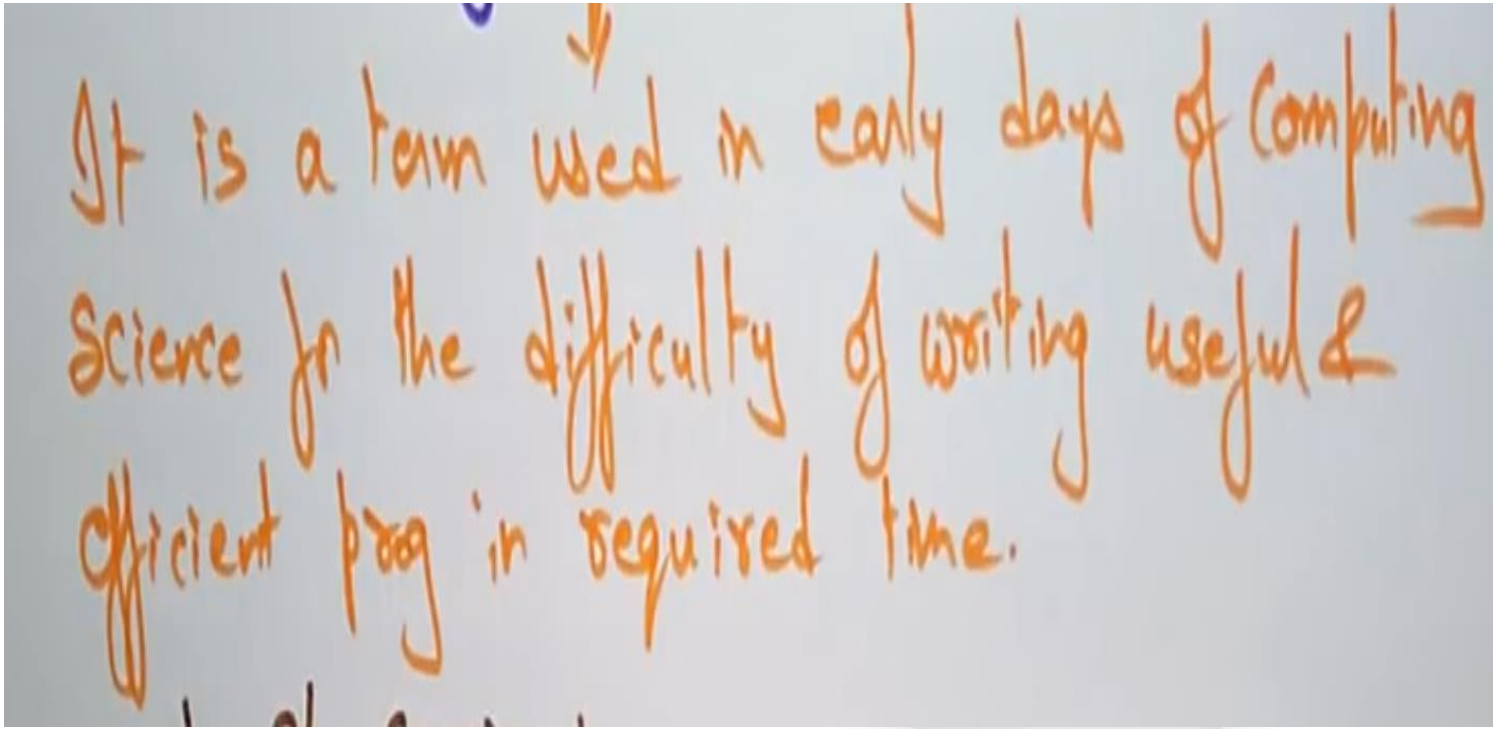
National infrastructures and utilities are controlled by systems Industrial manufacturing and distribution is completely computerized by Financial, management, decision making or control systems.

They are not constrained by the properties of materials, nor are they governed by physical laws or by manufacturing processes. This simplifies software engineering, as there are no natural limits to the potential of software. Complex, Difficult to understand Hardly to change.

Simple systems to world wide IS

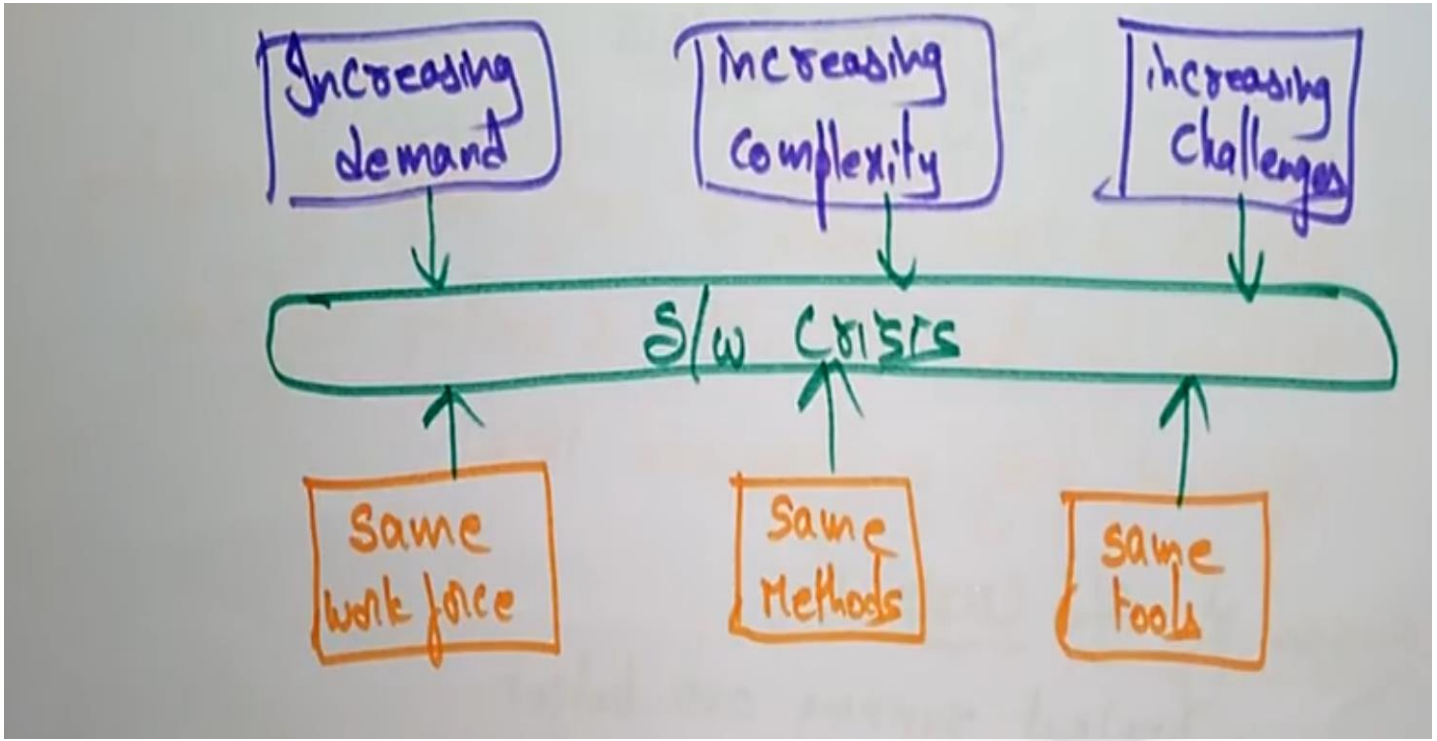


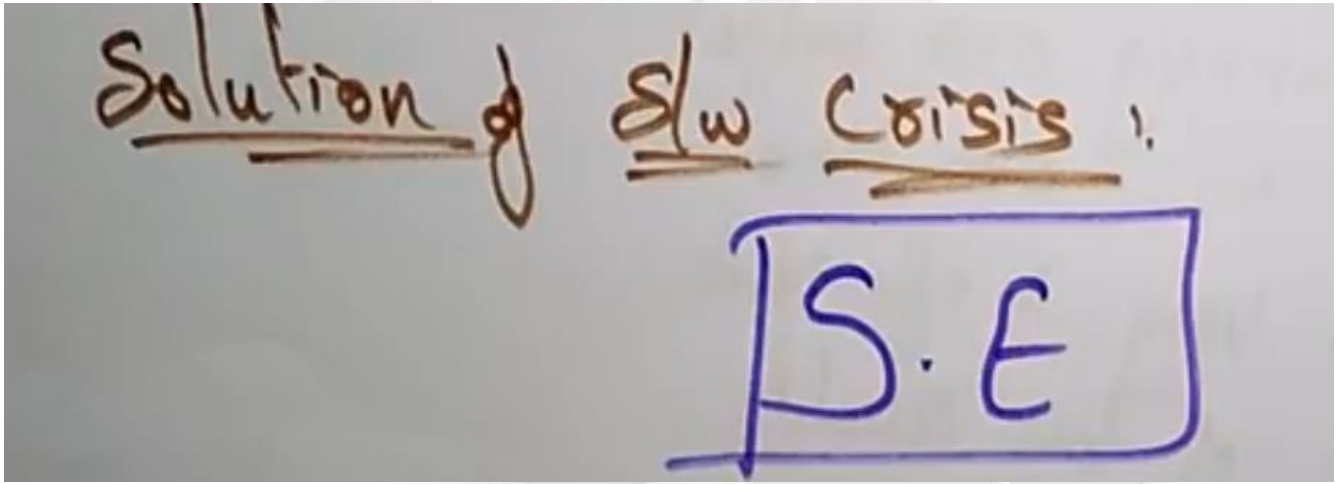
Software \ Crises



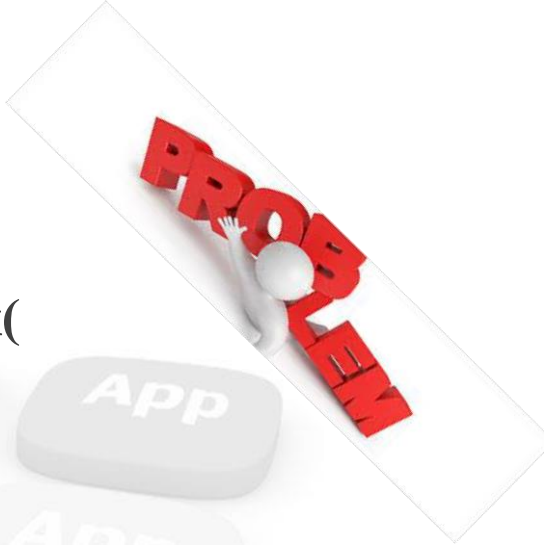
Causes of S/w Crisis :

- project running over budget
- " " ova time
- S/w was very inefficient
- " " of low Quality
- S/w often didn't meet requirements





Software Projects going wrong "failures?"



1. **Systems complexity. (Quick-Large-Complex(**
2. **Failures in applying the methods.**
3. **Evaluation.**

New Methods have to be Developed.

Education and Training for solving this problem.

Without the Software Engineers World will be Difficult.

Software Engineering Importance

Software engineering is important for two reasons:

1.More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.

2. It is usually cheaper, in the long run, to use software engineering methods and techniques for professional software systems rather than just write programs as a personal programming project.

Failure to use software engineering method leads to higher costs for testing, quality assurance, and long-term maintenance.

Professional software development

Lots of people write programs. People in business write spreadsheet programs to simplify their jobs; scientists and engineers write programs to process their experimental data; hobbyists write programs for their own interest and enjoyment.

However, most software development is a professional activity in which software is developed for business purposes, for inclusion in other devices, or as software products such as information systems and computer-aided design systems.

Software engineering is intended to support professional software development rather than individual programming.

It includes techniques that support program specification, design, and evolution, none of which are normally relevant for personal software development.

Professional software development

Many people think that software is simply another word for computer programs.

However, when we are talking about software engineering, software is not just the programs themselves but also all associated documentation, libraries, support web-sites, and configuration data that are needed to make these programs useful.

A professionally developed software system is often more than a single program.

A system may consist of several separate programs and configuration files that are used to set up these programs.

It may include system documentation, which describes the structure of the system, user documentation, which explains how to use the system, and websites for users to download recent product information.

Professional software development

This is one of the important differences between professional and amateur software development.

If you are writing a program for yourself, no one else will use it and you don't have to worry about writing program guides, documenting the program design, and so on.

However, if you are writing software that other people will use and other engineers will change, then you usually have to provide additional information as well as the code of the program.

Software Engineering

Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use. In this definition, there are two key phrases:

Engineering discipline Engineers make things work. They apply theories, methods, and tools where these are appropriate. However, they use them selectively and always try to discover solutions to problems even when there are no applicable theories and methods. Engineers also recognize that they must work within organizational and financial constraints, and they must look for solutions within these constraints.

All aspects of software production Software engineering is not just concerned with the technical processes of software development. It also includes activities such as software project management and the development of tools, methods, and theories to support software development.

Software Engineering

Engineering is about getting results of the required quality within schedule and budget. This often involves making compromises—engineers cannot be perfectionists. People writing programs for themselves, however, can spend as much time as they wish on the program development.

In general, software engineers adopt a systematic and organized approach to their work, as this is often the most effective way to produce high-quality software.

However, engineering is all about selecting the most appropriate method for a set of circumstances, so a more creative, less formal approach to development may be the right one for some kinds of software.

A more flexible software process that accommodates rapid change is particularly appropriate for the development of interactive web-based systems and mobile apps, which require a blend of software and graphical design skills.

Software Engineering (Computer Science and Systems Engineering(

Software engineering is related to both computer science and systems engineering.

1.Computer science is concerned with the theories and methods that underlie computers and software systems, whereas software engineering is concerned with the practical problems of producing software.

Some knowledge of computer science is essential for software engineers in the same way that some knowledge of physics is essential for electrical engineers. Computer science theory, however, is often most applicable to relatively small programs. Elegant theories of computer science are rarely relevant to large, complex problems that require a software solution.

2. System engineering is concerned with all aspects of the development and evolution of complex systems where software plays a major role. System engineering is therefore concerned with hardware development, policy and process design, and system deployment, as well as software engineering.

System engineers are involved in specifying the system, defining its overall architecture, and then integrating the different parts to create the finished system.

Software Engineering Diversity

Software engineering is a systematic approach to the production of software that takes into account practical cost, schedule, and dependability issues, as well as the needs of software customers and producers.

The specific methods, tools, and techniques used depend on the organization developing the software, the type of software, and the people involved in the development process.

There are no universal software engineering methods that are suitable for all systems and all companies.

Rather, a diverse set of software engineering methods and tools has evolved over the past 50years.

Software Engineering Diversity

Stand-alone applications These are application systems that run on a personal computer or apps that run on a mobile device. They include all necessary functionality and may not need to be connected to a network. Examples of such applications are office applications on a PC, CAD programs, photo manipulation software, travel apps, productivity apps, and so on.

Interactive transaction-based applications These are applications that execute on a remote computer and that are accessed by users from their own computers, phones, or tablets. Obviously, these include web applications such as e-commerce applications where you interact with a remote system to buy goods and services.

Embedded control systems These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system. Examples of embedded systems include the software in a mobile (cell) phone.

Software Engineering Diversity

Batch processing systems These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs. Examples of batch systems are periodic billing systems, such as phone billing systems, and salary payment systems.

Entertainment systems These are systems for personal use that are intended to entertain the user. Most of these systems are games of one kind or another, which may run on special-purpose console hardware. The quality of the user interaction offered is the most important distinguishing characteristic of entertainment systems.

Systems for modeling and simulation These are systems that are developed by scientists and engineers to model physical processes or situations, which include many separate, interacting objects. These are often computationally intensive and require high-performance parallel systems for execution.

Software Engineering Diversity

Data collection and analysis systems Data collection systems are systems that collect data from their environment and send that data to other systems for processing. The software may have to interact with sensors and often is installed in a hostile environment such as inside an engine or in a remote location. “Big data” analysis may involve cloud-based systems carrying out statistical analysis and looking for relationships in the collected data.

Systems of systems These are systems, used in enterprises and other large organizations, that are composed of a number of other software systems. Some of these may be generic software products, such as an ERP system. Other systems in the assembly may be specially written for that environment.

Software Life Cycle (SWLF)

* SWLF is the concept of SE and its consists of stages.

*General process model: Descriptive and Diagrammatic model of SWLF.

Analysis->Design->Implementation->Maintenance->Planning

Development team are responsible for SWLC suitable model based on (time, specification, budget and abilities).

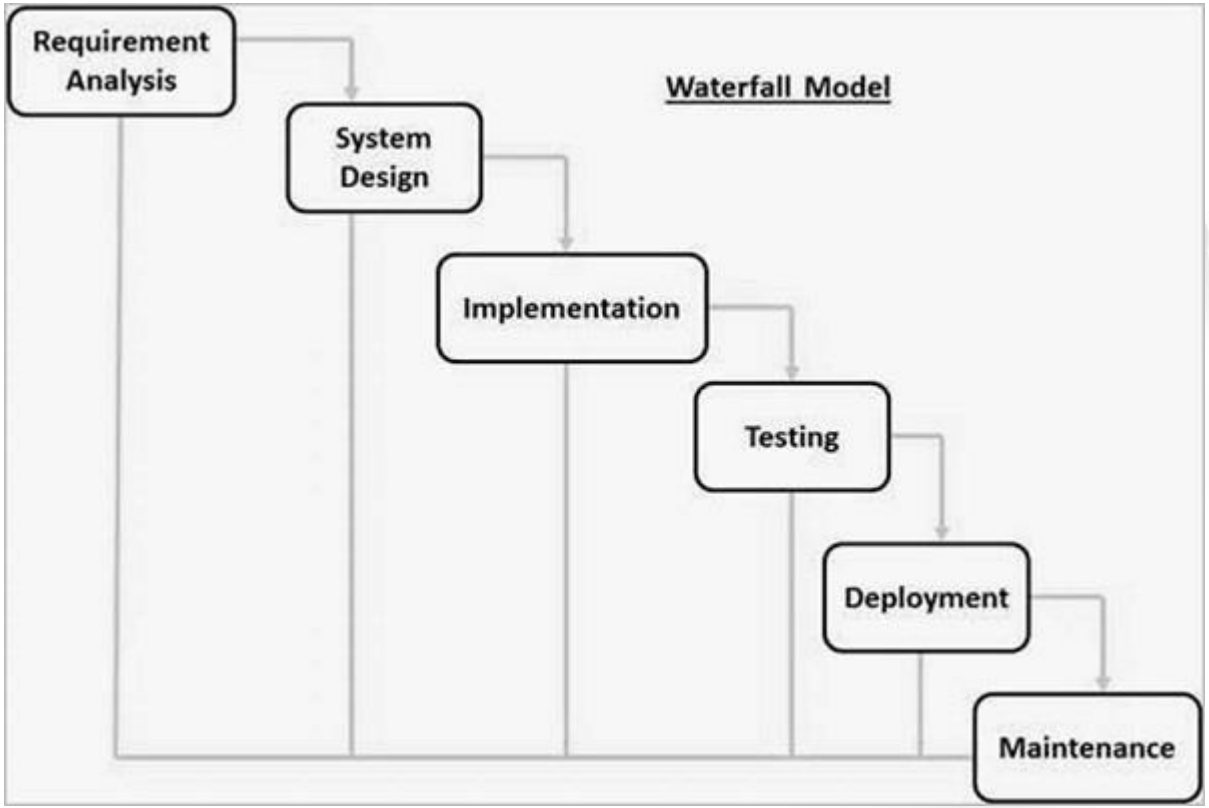
If they don't follow any model so this is mean they do their own personal way.

Following SWLF models gives a clear understanding for team and project manager of what is happening what to do what should do.

In fact there are many models (with advantages and disadvantages) some of these models categorized as methodologies

SDLC - Waterfall Model SDLC - Iterative Model SDLC - Spiral Model SDLC – V Model SDLC - Big Bang Model SDLC - Agile Model SDLC - RAD Model SDLC - Software Prototype

WaterFall



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

1. Requirements analysis and definition The system's services.
2. System and software design The systems design process allocates the requirements to either hardware or software systems.
3. Implementation and unit testing During this stage, the software design is realized as a set of programs or program units.
4. Integration and system testing The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met.
5. Operation and maintenance Normally, this is the longest life-cycle phase. The system is installed and put into practical use

Embedded systems - Critical systems - Large software systems that are part of broader engineering systems developed by several partner companies.

Waterfall Model - Application

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are −

Requirements are very well documented, clear and fixed.
Product definition is stable.
Technology is understood and is not dynamic.
There are no ambiguous requirements.
Ample resources with required expertise are available to support the product.
The project is short.

Waterfall Model - Advantages

The advantages of waterfall development are that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

Simple and easy to understand and use

Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.

Phases are processed and completed one at a time.

Works well for smaller projects where requirements are very well understood.

Clearly defined stages.

Well understood milestones.

Easy to arrange tasks.

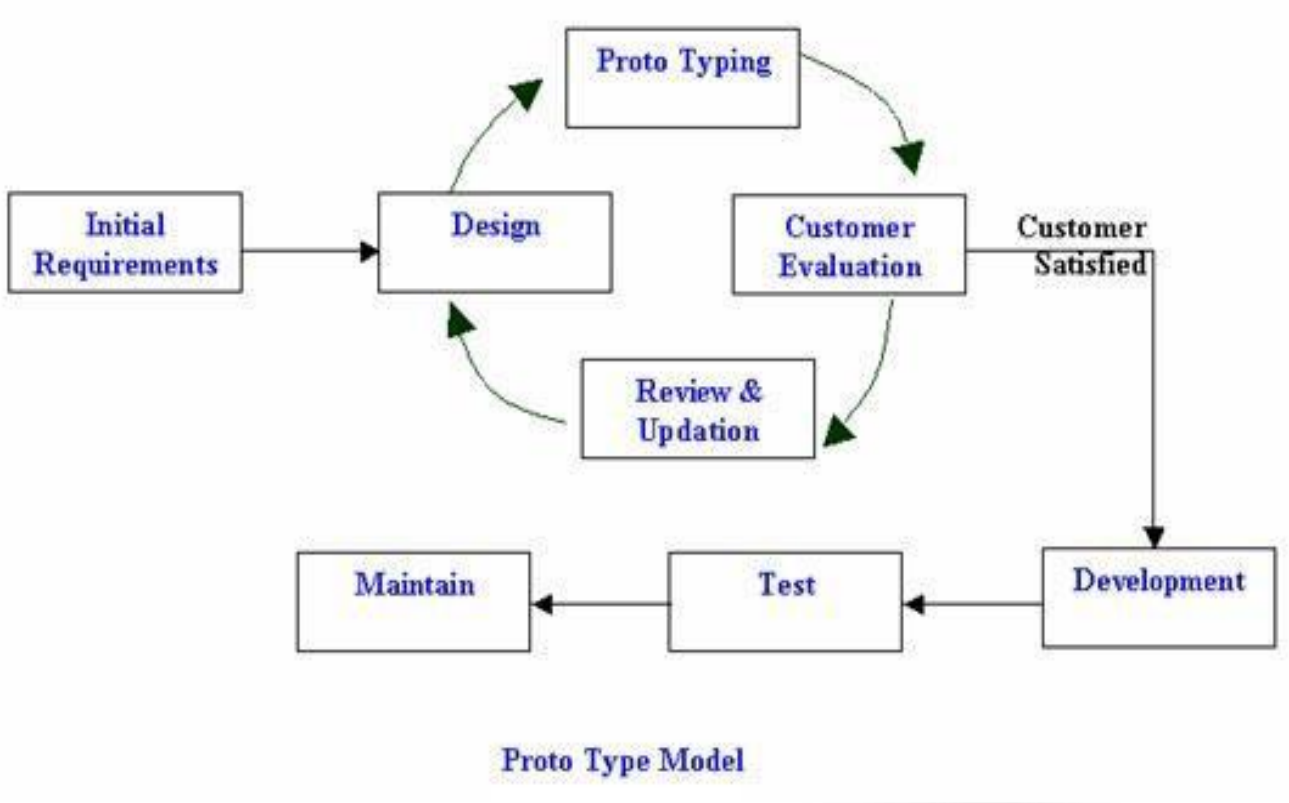
Process and results are well documented.

Waterfall Model - Disadvantages

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

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. So, risk and uncertainty is high with this process model.
It is difficult to measure progress within stages.
Cannot accommodate changing requirements.
Adjusting scope during the life cycle can end a project.
Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Prototyping



Proto Type Model

Prototyping

The Software Prototyping refers to building software application prototypes which displays the functionality of the product under development, but may not actually hold the exact logic of the original software.

Software prototyping is becoming very popular as a software development model, as it enables to understand customer requirements at an early stage of development.

It helps get valuable feedback from the customer and helps software designers and developers understand about what exactly is expected from the product under development.

Prototyping Advantages

The advantages of the Prototyping Model are as follows −

Increased user involvement in the product even before its implementation.
Since a working model of the system is displayed, the users get a better understanding of the system being developed.
Reduces time and cost as the defects can be detected much earlier.
Quicker user feedback is available leading to better solutions.
Missing functionality can be identified easily.
Confusing or difficult functions can be identified.

Prototyping Disadvantages

The Disadvantages of the Prototyping Model are as follows −

Risk of insufficient requirement analysis owing to too much dependency on the prototype.
Users may get confused in the prototypes and actual systems.
Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
Developers may try to reuse the existing prototypes to build the actual system, even when it is not technically feasible.

Iterative Incremental

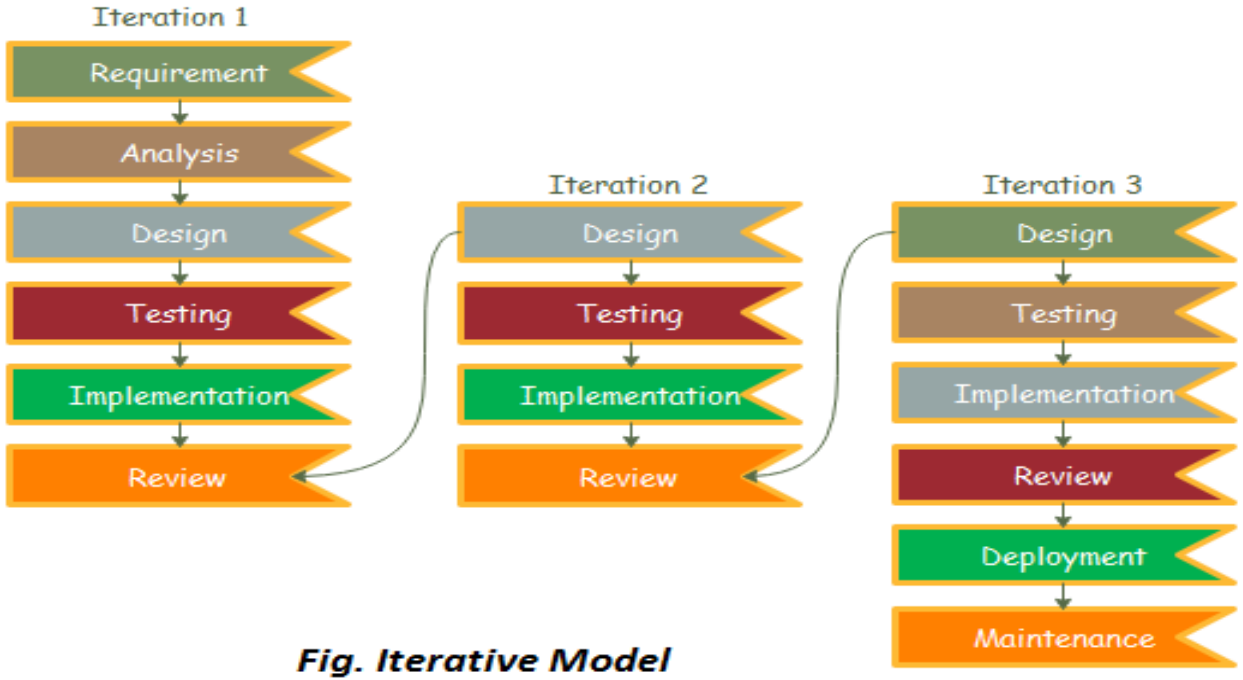
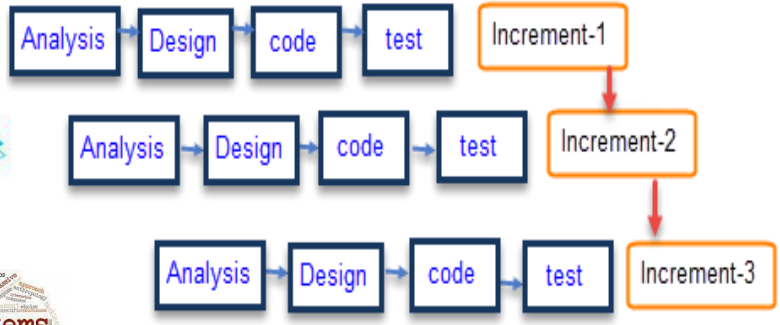
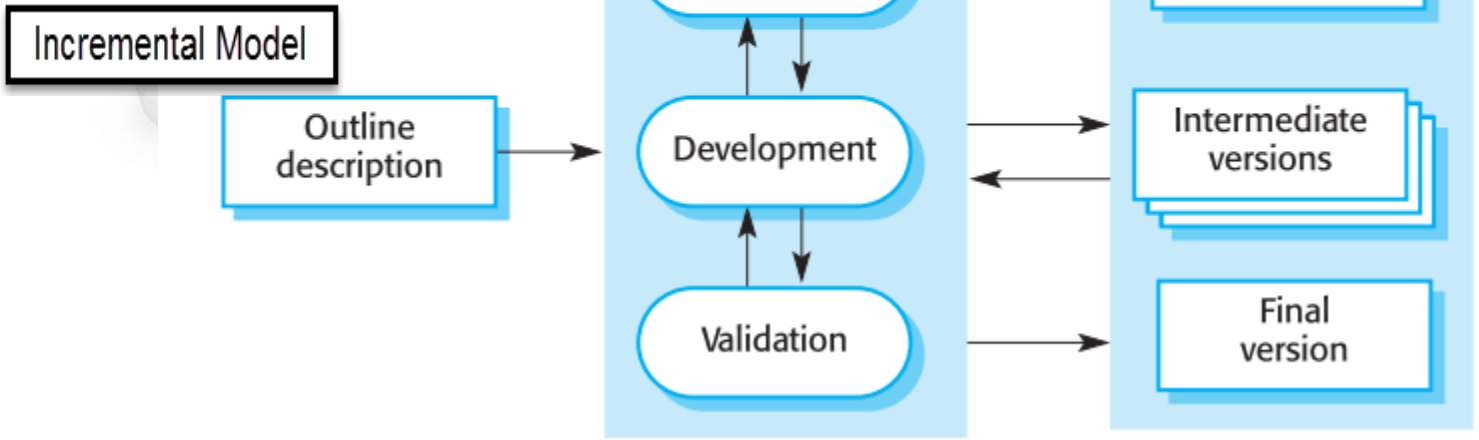


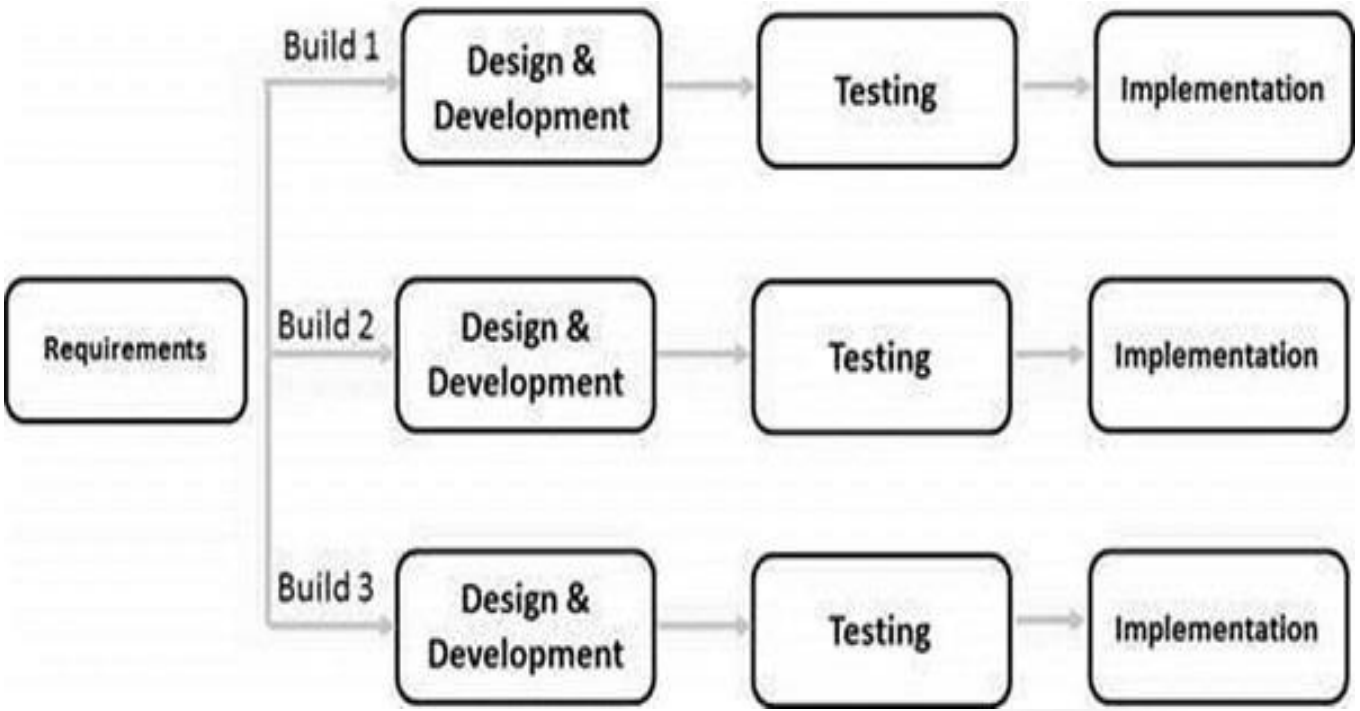
Fig. Iterative Model

Iterative Incremental





Iterative Incremental



Iterative Incremental- Application

Like other SDLC models, Iterative and incremental development has some specific applications in the software industry. This model is most often used in the following scenarios −

Requirements of the complete system are clearly defined and understood.

Major requirements must be defined; however, some functionalities or requested enhancements may evolve with time.

There is a time to the market constraint.

A new technology is being used and is being learnt by the development team while working on the project.

Resources with needed skill sets are not available and are planned to be used on contract basis for specific iterations.

There are some high-risk features and goals which may change in the future.

Iterative Incremental Model - Advantages

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.

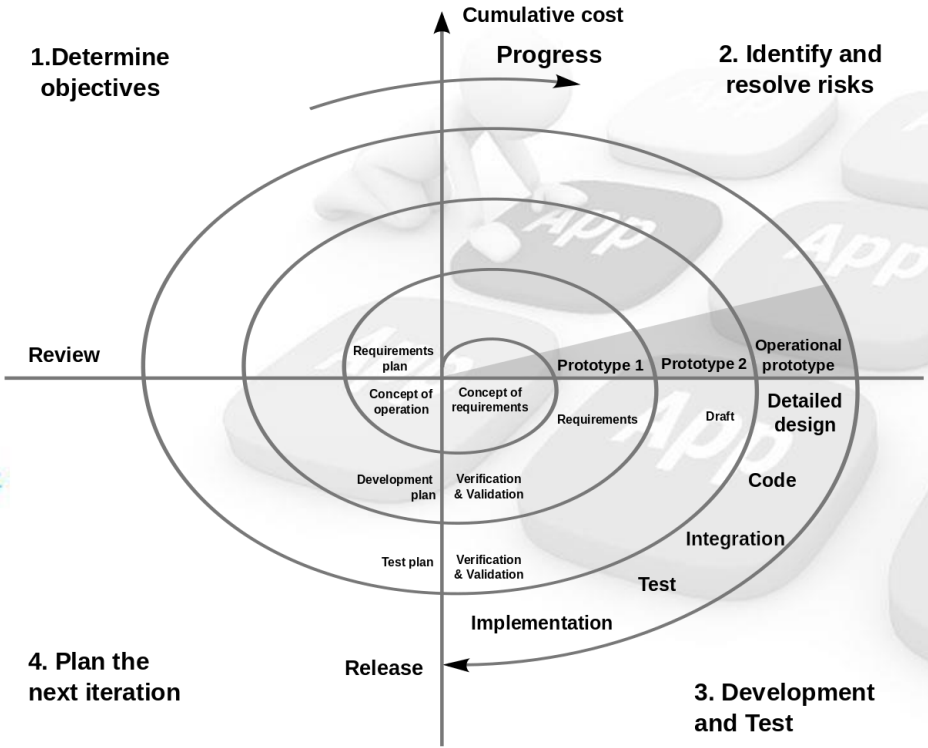
Some working functionality can be developed quickly and early in the life cycle.
Results are obtained early and periodically.
Parallel development can be planned.
Progress can be measured.
Less costly to change the scope/requirements.
Testing and debugging during smaller iteration is easy.
Risks are identified and resolved during iteration; and each iteration is an easily managed milestone.
Easier to manage risk - High risk part is done first.
With every increment, operational product is delivered.
Risk analysis is better.
It supports changing requirements.

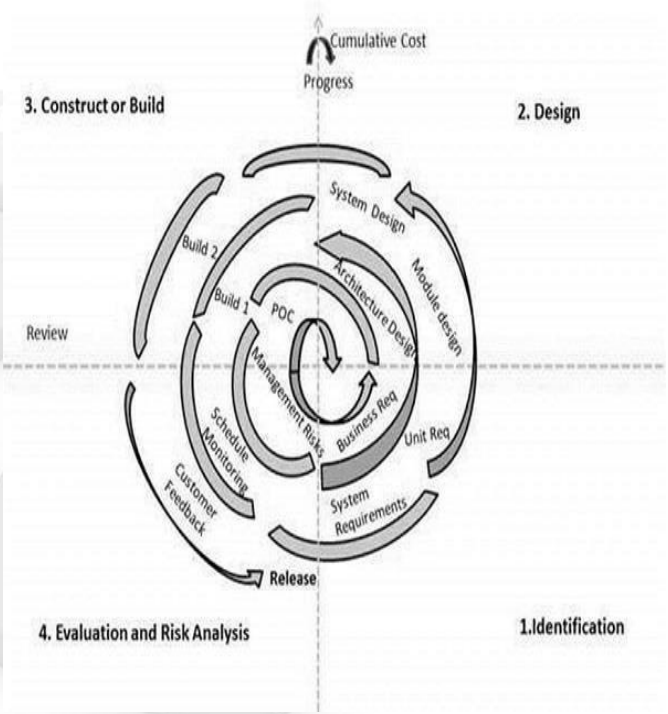
Iterative Incremental Model - Disadvantages

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.

More resources may be required.
Although cost of change is lesser, but it is not very suitable for changing requirements.
More management attention is required.
System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle.
Defining increments may require definition of the complete system.
Not suitable for smaller projects.
Management complexity is more.
End of project may not be known which is a risk.
Highly skilled resources are required for risk analysis.
Projects progress is highly dependent upon the risk analysis phase.

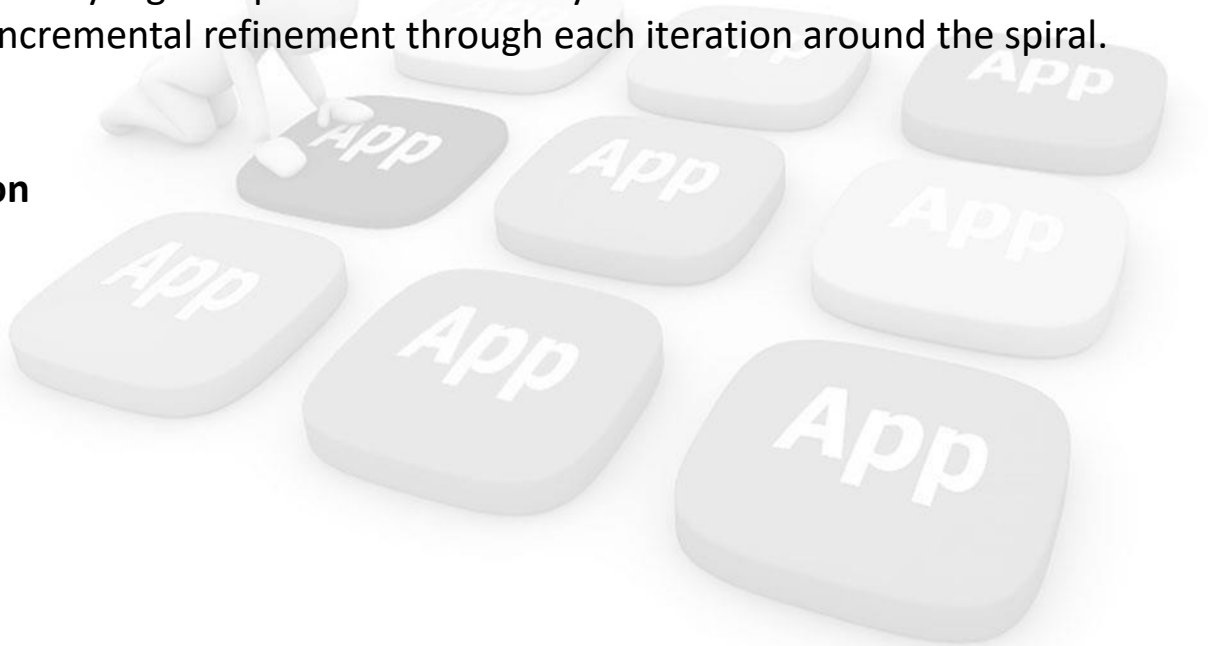
Spiral





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

Identification
Design
Construct
Evaluation



Spiral Model Application

The Spiral Model is widely used in the software industry as it is in sync with the natural development process of any product, i.e. learning with maturity which involves minimum risk for the customer as well as the development firms.

The following pointers explain the typical uses of a Spiral Model −

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

Spiral Model Advantages

The advantages of the Spiral SDLC Model are as follows −

Changing requirements can be accommodated.
Allows extensive use of prototypes.
Requirements can be captured more accurately.
Users see the system early.
Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

Spiral Model Disadvantages

The disadvantages of the Spiral SDLC Model are as follows −

Management is more complex.
End of the project may not be known early.
Not suitable for small or low risk projects and could be expensive for small projects.
Process is complex.
Spiral may go on indefinitely.
Large number of intermediate stages requires excessive documentation.