Unit 1

Software Engineering, it is a systematic approach to develop, operates, maintain and retire the software.

Software crisis

Software crisis was a term used in the early days of computing science. In which Software projects mostly failed to meet cost and time and requirements.

This crisis in several ways:

- Projects running over-budget.
- Projects running over-time.
- Software was of low quality.
- Software often did not meet requirements.
- Projects were unmanageable and code difficult to maintain.

The term "software crisis" was overcome by some attendees at the first NATO Software Engineering Conference in 1968 at Germany. Which gives the solution of this problem through software development life cycle?

Software Characteristics

1. Software is developed or engineered, not manufactured.

The difference is in their implementation part. They both differ in their coding part. So, it is said that software is not manufactured but it is developed or engineered.

2. Software doesn't wear out:

The hardware can wear out whereas software can't. In case of hardware we have a "bathtub" like curve, which is a curve that lies in between failure-rate and time. In this curve, in the starting time there is relatively high failure rate. But, after some period of time, defects get corrected and failure-rate drops to a steady-state for some time period. But, the failure-rate again rises due to the effects of rain, dust, temperature extreme and many other environment effects. The hardware begins to wear out.

But, the software is not responsible to the failure rate of hardware. The failure rate of software can be understood by the "idealized curve". In this type of curve the failure rate in the initial state is very high. But, the errors in the software get corrected and the curve flattens. so software does not "wear out".

3. Reusability of component:

Software can be developed. You don't need to start writing the code from the scratch. But you can use already developed software to develop another one, e.g. a graphical user interface can be developed by the software which are developed prior to your new software. You don't need to think over it that how it can be developed. You just use the already developed code for it.

4. Software is flexible:

e is said to be "flexible" because, software can be developed for doing anything and any type of n. Software can be developed for solving any type of problem and can also be changed if the ements change.

ware quality attributes

ocus on

Design quality

Run time quality

System quality

User quality

Design quality

eusability defines the capability for components and subsystems to be suitable for use in other applications. eusability minimizes the duplication of components and also the implementation time.

vailability defines the ratio of time that the system is functional and working. It can be measured as a ercentage of the total system downtime over a predefined period. Availability will be affected by system rrors, infrastructure problems, malicious attacks, and system load.

nteroperability is the ability of a system or different systems to operate successfully by communicating and xchanging information with other.

Maintainability is the ability of the system to changes. These changes could impact components, services, eatures, and interfaces when adding or changing the functionality,

2. Run time quality

Performance is ahint of the responsiveness of a system to execute any action within a given time interval. It an be measured in terms of latency or throughput. Latency is the time taken to respond to any event. Throughput is the number of events that take place within a given amount of time.

Reliability is the ability of a system to remain operational over time. Reliability is measured as system will not fail to perform its intended functions over a specified time interval.

Manageability defines how easy it is for system administrators to manage the application,

Security, A secure system aims to protect assets and prevent unauthorized modification of information.

3. System quality

Testability is a measure of how easy it is to create test criteria for the system and its components. 4. User quality

Usability defines how well the application meets the requirements of the user.

structure of SE

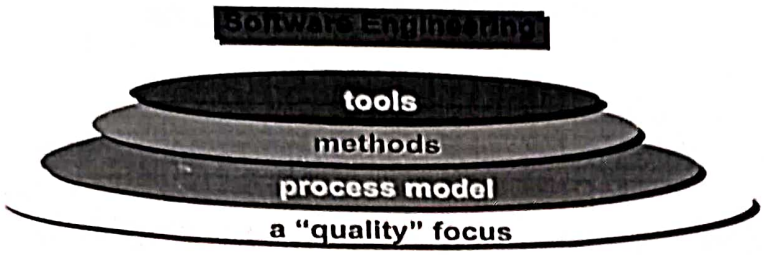
/are engineering is a systematic approach to development, operation and maintenance and retire of /ware.

/r

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

Software engineering is concerned with building good and reliable software. Any branch of engineering uses the following for designing and building a product:

- Process
- Methods
- Tools



Software engineering provides a set of methods, tools and procedures for development of quality software within the constraints of cost and time.

A software process is ***a set of activities and associated results***, which produces a software product. Software engineers carry out the activities. The common activities cabe categorized into:

- ***Software specification*** - the functionality of the software and constraints on its operation must be defined.
- ***Software development*** – the software to meet the specification must be produced.
- ***Software validation*** – the software must be validated to ensure that it does what the customer wants.
- ***Software evolution*** – the software must be evolved to meet the changing requirements of customers.

Methods provide the rules, steps and techniques for software engineering tasks, such as planning and estimation, requirement analysis, data structure, algorithm procedure, coding, testing and maintenance of s/w. There are mainly two approaches to software engineering, namely, (i) function-oriented approach and (ii) object-oriented approach.

Function-oriented approach is the traditional approach of software development. In which, a system is to perform some set of functions. Firstly, a system is broken down into subsystems. Each basic function, the

nding lower-level functions are determined and accordingly subsystems are broken down into further ems. Finally, each lower-level function is studied in detail to determine how it is performed and the ponding software component is designed accordingly.

ject-oriented approach is a modern approach. It supplements function-oriented method by trying to identify sic objects, their properties and behaviors.

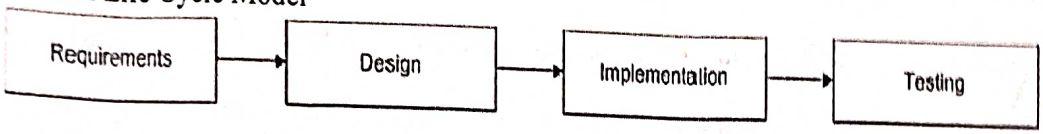
Tools provide support for methods and process. They consist of various graphical representations, tables and charts that help in analysis of software requirements and its design. Data Flow Diagram (DFD), Data Dictionary, Entity-Relation (ER) diagram, Decision Table, Structured Charts etc. are some of the important tools used in the function-oriented approach. In the object-oriented approach various tools are integrated into a single unified language called "Unified Modeling Language" (UML). This language is now widely used by professionals for software development.

These software tools are called Computer-Aided Software Engineering (CASE) tools.

The General Model

oftware life cycle models describe phases of the software cycle and the order in which those phases are executed. There are tons of models, and many companies adopt their own, but all have very similar patterns. The general, basic model is shown below:

General Life Cycle Model



Each phase produces deliverables required by the next phase in the life cycle. Requirements are translated into design. Code is produced during implementation that is driven by the design. Testing verifies the deliverable of the implementation phase against requirements.

Requirements

Software requirements are gathered in this phase. . Meetings with users are held in order to determine the requirements. Who is going to use the system? How will they use the system? What data should be input into e system? What data should be output by the system? These are general questions that get answered during a requirements gathering phase, means developer all information fetched by the user.

Design

The software design is produced from the results of the requirements phase. Architects have the ball in their court during this phase and this is the phase in which their focus lies. This is where the details on how the system will work is produced. Architecture, including hardware and software, communication, software design (UML is produced here) are all part of the deliverables of a design phase.

Implementation

Code is produced from the deliverables of the design phase during implementation, and this is the longest phase of the software development life cycle. For a developer, this is the main focus of the life cycle because this is where the code is produced. Implementation my overlap with both the design and testing phases. Many tools

ASE tools) to actually automate the production of code using information gathered and produced the design phase.

ing

uring testing, the implementation is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. Unit tests and system/acceptance tests are done during this phase. Unit tests act on a specific component of the system, while system tests act on the system as a whole.

Maintenance: when the customers start 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 process models

There are many process models

Waterfall Model

This is the most common and classic of life cycle models, 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 in its entirety before the next phase can begin. 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, if requirement comes after the completed the phases then it is not ready for adding basically it is applicable where all documents are fulfilled or enhance the existing software.

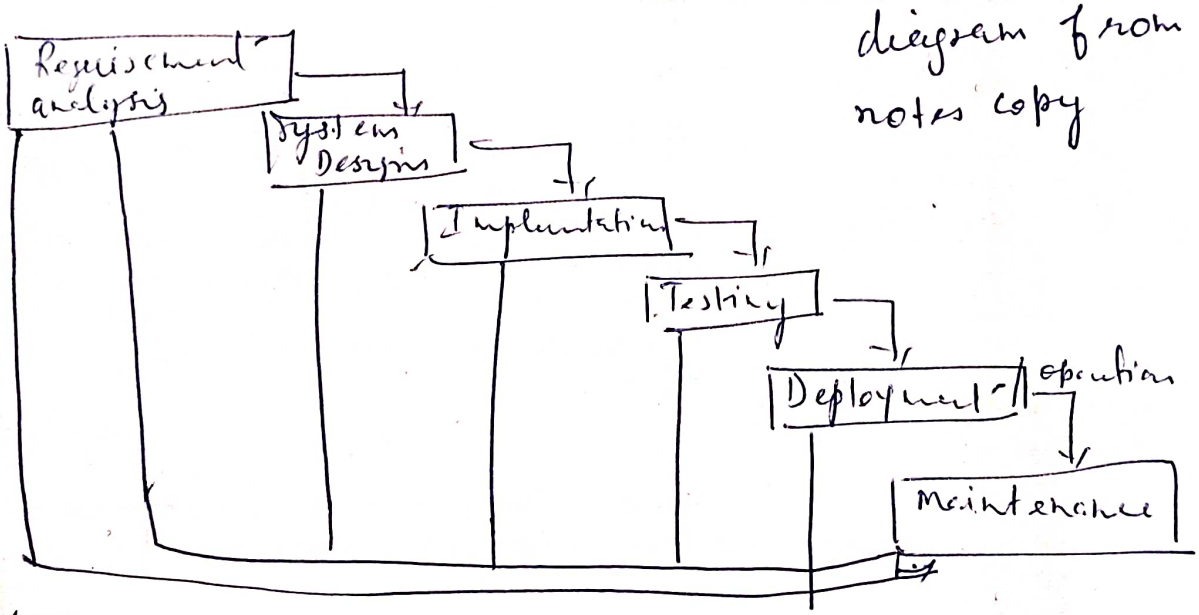


diagram from notes copy

Advantages

- Simple and easy to use.
- It is segmental approach.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.

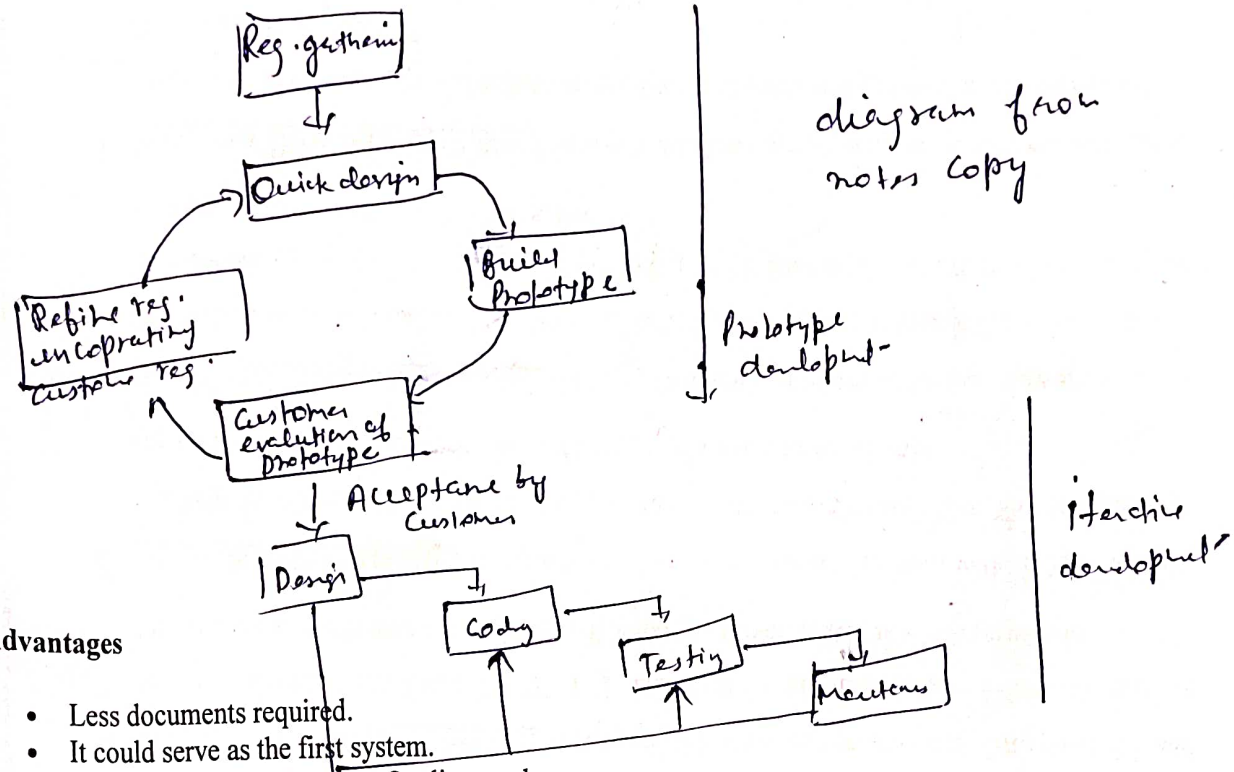
Disadvantages

- High amounts of risk and uncertainty.
- Poor model for complex and large projects.
- Poor model where requirements are at a moderate to high risk of changing.
- Cannot follows the go back process means requirements are in freeze form.

Prototyping model

It is developed to overcome the weaknesses of the waterfall model. It starts with an initial planning and ends with deployment with the cyclic interactions in between.and it is applicable where user does not know the entire requirement at the initial phase. So it is iterative in nature.

It begins with requirements gathering, in this model developer and user both are communicate together then developer ready the quick design of prototype and send to user end, user take the test drive then give the feedback if user is satisfied than ok otherwise it follows the go back process until user is not satisfied.



Advantages

- Less documents required.
- It could serve as the first system.
- Fully satisfaction of user so Quality good.
- Provide better interaction between user and developer.

Disadvantages

- In this model we cannot estimate the cost at initial level.
- Developer and user both do not know what the look of the software at last.

Spiral Model

The spiral model is similar to the incremental model, with more emphases on risk analysis. The spiral model has four phases: Planning, Risk Analysis, developed and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spirals, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spiral builds on the baseline spiral.

ents are gathered during the planning phase.

risk analysis phase, a process is undertaken to identify risk and alternate solutions. A prototype is uced at the end of the risk analysis phase.

oftware is produced in the developed phase, along with testing at the end of the phase.

The evaluation phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

In the spiral model, the angular component represents progress, and the radius of the spiral represents cost.

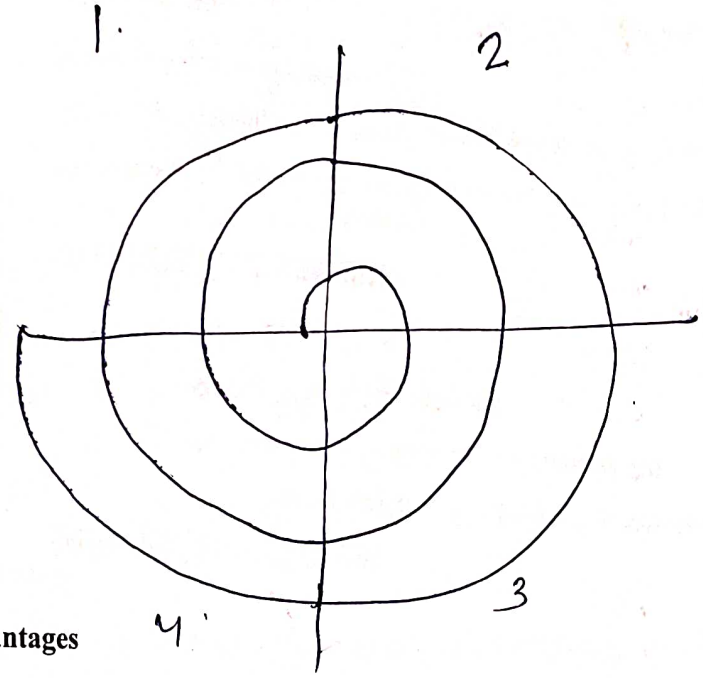


diagram from notes copy

1. Objectives determination and identify alternative sol.
2. Identify and resolve Risks
3. Develop next version of the product.
4. Review & plan for the next phase.

Advantages

- Follow the risk analysis approach.
- Good for large and mission-critical projects.
- Less documents need.

Disadvantages

- It is very costly model than others.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

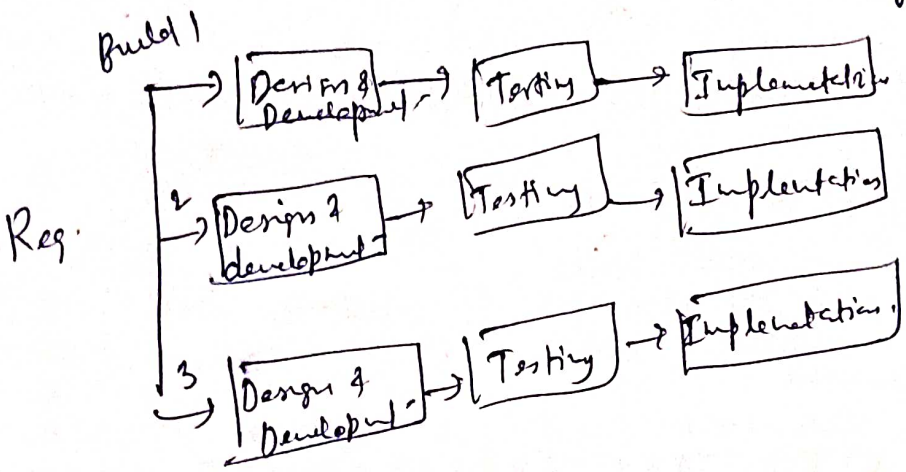
Incremental Model

The incremental model is a sensitive approach to the waterfall model. Multiple development cycles take place here, making the life cycle a "multi-waterfall" cycle. Cycles are divided up into smaller, more easily managed iterations. Each iteration passes through the requirements, design, implementation and testing phases.

ng version of software is produced during the first iteration, so you have working software early on the software life cycle. Subsequent iterations build on the initial software produced during the first ion.

is used for large system which built in part by part segment. Also can be used in system has separated components, for example, ERP system. Which we can start with budget module as first iteration and then we can start with inventory module and so forth.

diagram from notes copy



Advantages

- Generates working software quickly and early during the software life cycle.
- More flexible.
- Easier to test and debug during a smaller iteration.
- Easier to manage risk because risky pieces are identified and handled during its iteration.

Disadvantages

- Each phase of an iteration is rigid and do not overlap each other.
- Problems may arise pertaining to system architecture because not all requirements are gathered up front for the entire software life cycle.

Q) SOME CHARACTERISTICS OF SOFTWARE INCLUDES:-

1) Software is developed or engineer.
2) Most of software is custom build rather than assemble from existing component.
3) Computer program and associated documentation.
4) Easy to modified.
5) Easy to reproduce.
6) Software product may be developed for a particular customer or for the general market.

Q) DIFFERENCE BETWEEN PROGRAM AND SOFTWARE.

PROGRAM	SOFTWARE
1) Small in size. 2) Authors himself is user-soul. 3) Single developer. 4) Adopt development. 5) Lack proper interface. 6) Large proper documentation.	1) Large in size. 2) Large number. 3) Team developer. 4) Systematic development. 5) Well define interface. 6) Well documented.

DEFINITION OF SOFTWARE: - it is systematic approach to the development, operation, maintenance and retirement of software. It is the application of computer science along with mathematics and ergative science. In the current scenario the S.E has a specific importance for making particular software.

D (Rapid Application Development) model is based on prototyping and iterative development with no
e planning involved.

id Application Development focuses on gathering customer requirements through workshops or focus
ups, early testing of the prototypes by the customer using iterative concept, reuse of the existing prototypes
components), continuous integration and rapid delivery.

Rapid application development is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product.

In the RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery.

The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable.

RAD Model Design

Following are the various phases of the RAD Model −

Business Modeling

The business model for the product under development is designed in terms of flow of information and the distribution of information between various business channels. A complete business analysis is performed to find the vital information for business, how it can be obtained, how and when is the information processed and what are the factors driving successful flow of information.

Data Modeling

The information gathered in the Business Modeling phase is reviewed and analyzed to form sets of data objects vital for the business. The attributes of all data sets is identified and defined. The relation between these data objects are established and defined in detail in relevance to the business model.

Process Modeling

The data object sets defined in the Data Modeling phase are converted to establish the business information flow needed to achieve specific business objectives as per the business model. The process model for any changes or enhancements to the data object sets is defined in this phase. Process descriptions for adding, deleting, retrieving or modifying a data object are given.

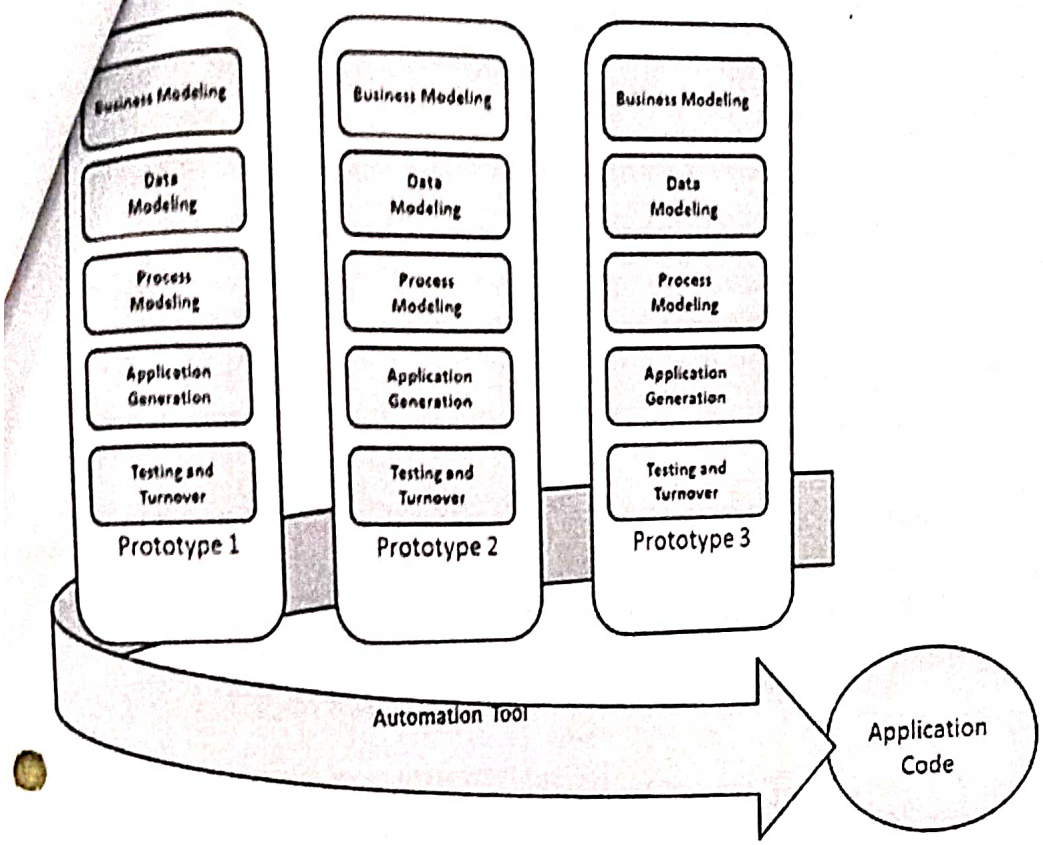
Application Generation

The actual system is built and coding is done by using automation tools to convert process and data models into actual prototypes.

Testing and Turnover

The overall testing time is reduced in the RAD model as the prototypes are independently tested during every iteration. However, the data flow and the interfaces between all the components need to be thoroughly tested with complete test coverage. Since most of the programming components have already been tested, it reduces the risk of any major issues.

owing illustration describes the RAD Model in detail.



The advantages of the RAD Model are as follows –

- Changing requirements can be accommodated.
- Progress can be measured.
- Iteration time can be short with use of powerful RAD tools.
- Productivity with fewer people in a short time.
- Reduced development time.
- Increases reusability of components.
- Quick initial reviews occur.
- Encourages customer feedback.
- Integration from very beginning solves a lot of integration issues.

The disadvantages of the RAD Model are as follows –

- Dependency on technically strong team members for identifying business requirements.
- Only system that can be modularized can be built using RAD.
- Requires highly skilled developers/designers.
- High dependency on modeling skills.
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.
- Management complexity is more.
- Suitable for systems that are component based and scalable.
- Requires user involvement throughout the life cycle.

Suitable for project requiring shorter development times.

olutionary Process Models

Evolutionary models are iterative type models.

They allow developing more complete versions of the software.

Following are the evolutionary process models.

1. The prototyping model
2. The spiral model
3. Concurrent development model

Software metrics is a standard of measure that contains many activities which involve some degree of measurement. It can be classified into three categories: product metrics, process metrics, and project metrics.

- **Product metrics** describe the characteristics of the product such as size, complexity, design features, performance, and quality level.
- **Process metrics** can be used to improve software development and maintenance. Examples include the effectiveness of defect removal during development, the pattern of testing defect arrival, and the response time of the fix process.
- **Project metrics** describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.

Some metrics belong to multiple categories. For example, the in-process quality metrics of a project are both process metrics and project metrics.

Scope of Software Metrics

Software metrics contains many activities which include the following –

- Cost and effort estimation
- Productivity measures and model
- Data collection
- Quantity models and measures
- Reliability models
- Performance and evaluation models
- Structural and complexity metrics
- Capability – maturity assessment
- Management by metrics
- Evaluation of methods and tools

Software measurement is a diverse collection of these activities that range from models predicting software project costs at a specific stage to measures of program structure.

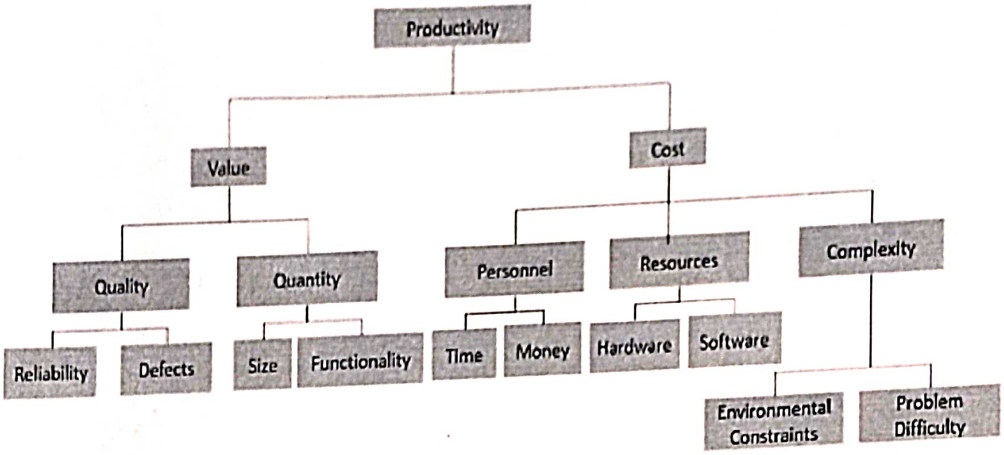
Cost and Effort Estimation

s expressed as a function of one or more variables such as the size of the program, the bility of the developers and the level of reuse. Cost and effort estimation models have been posed to predict the project cost during early phases in the software life cycle. The different dels proposed are –

- Boehm's COCOMO model
- Putnam's slim model
- Albrecht's function point model

Productivity Model and Measures

Productivity can be considered as a function of the value and the cost. Each can be decomposed into different measurable size, functionality, time, money, etc. Different possible components of a productivity model can be expressed in the following diagram.



Q) Some characteristics of software

includes:-

1) Software is developed or engineer.

2) Most of software is custom build rather than assemble from

existing component.

3) Computer program and associated documentation.

4) Easy to modified.

5) Easy to reproduce.

6) Software product may be developed for a particular customer or

for the general market.

Unit 2

SOFTWARE REQUIREMENT CHARACTERISTICS Gathering software requirements are the foundation of the entire software development project. Hence they must be clear, correct and well-defined. A complete Software Requirement Specification must be:

• Clear • Correct • Consistent • Coherent • Comprehensible • Modifiable • Verifiable • Prioritized • Unambiguous • Traceable • Credible source

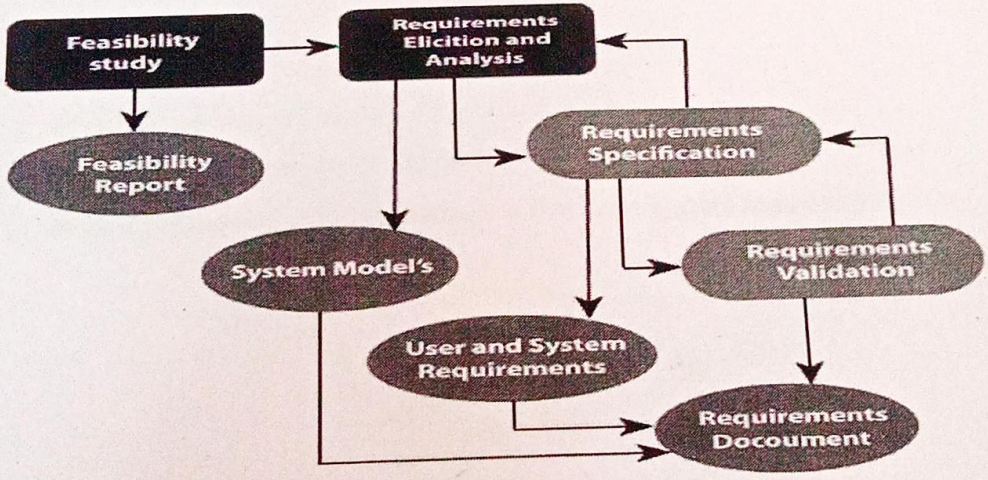
Requirement Engineering

The process to gather the software requirements from client, analyze and document them is known as requirement engineering.

Requirement Engineering Process

It is a four-step process, which includes -

1. Feasibility Study
2. Requirement Elicitation and Analysis
3. Software Requirement Specification
4. Software Requirement Validation
5. Software Requirement Management



Requirement Engineering Process

1. Feasibility Study:

The objective behind the feasibility study is to create the reasons for developing the software that is acceptable to users, flexible to change and conformable to established standards.

Types of Feasibility:

1. **Technical Feasibility** - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.
2. **Operational Feasibility** - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.
3. **Economic Feasibility** - Economic feasibility decides whether the necessary software can generate financial profits for an organization.

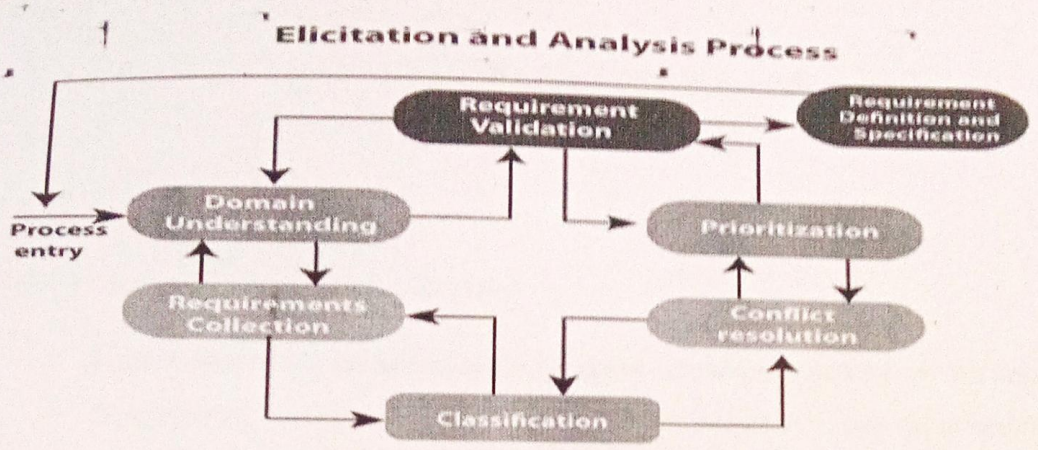
2. Requirement Elicitation and Analysis:

This is also known as the **gathering of requirements**. Here, requirements are identified with the help of customers and existing systems processes, if available.

Analysis of requirements starts with requirement elicitation. The requirements are analyzed to identify inconsistencies, defects, omission, etc. We describe requirements in terms of relationships and also resolve conflicts if any.

Problems of Elicitation and Analysis

- Getting all, and only, the right people involved.
- Stakeholders often don't know what they want
- Stakeholders express requirements in their terms.
- Stakeholders may have conflicting requirements.
- Requirement change during the analysis process.
- Organizational and political factors may influence system requirements.



3. Software Requirement Specification:

Software requirement specification is a kind of document which is created by a software analyst after the requirements collected from the various sources - the requirement received by the customer written in ordinary language. It is the job of the analyst to write the requirement in technical language so that they can be understood and beneficial by the development team.

The models used at this stage include ER diagrams, data flow diagrams (DFDs), function decomposition diagrams (FDDs), data dictionaries, etc.

- **Data Flow Diagrams:** Data Flow Diagrams (DFDs) are used widely for modeling the requirements. DFD shows the flow of data through a system. The system may be a company, an organization, a set of procedures, a computer hardware system, a software system, or any combination of the preceding. The DFD is also known as a data flow graph or bubble chart.
- **Data Dictionaries:** Data Dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and terminologies.
- **Entity-Relationship Diagrams:** Another tool for requirement specification is the entity-relationship diagram, often called an "***E-R diagram***." It is a detailed logical representation of the data for the organization and uses three main constructs i.e. data entities, relationships, and their associated attributes.

4. Software Requirement Validation:

After requirement specifications developed, the requirements discussed in this document are validated. The user might demand illegal, impossible solution or experts may misinterpret the needs. Requirements can be the check against the following conditions -

- If they can practically implement
- If they are correct and as per the functionality and specially of software
- If there are any ambiguities
- If they are full
- If they can describe

Requirements Validation Techniques

- **Requirements reviews/inspections:** systematic manual analysis of the requirements.
- **Prototyping:** Using an executable model of the system to check requirements.
- **Test-case generation:** Developing tests for requirements to check testability.
- **Automated consistency analysis:** checking for the consistency of structured requirements descriptions.

Software Requirement Management:

Requirement management is the process of managing changing requirements during the requirements engineering process and system development.

How to Write an IEEE SRS Document

1. Define the Purpose with an Outline (Or Use an SRS Template)

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 Nonfunctional Requirements

2. Define your Product's Purpose

This introduction is very important as it sets expectations that we will hit throughout the SRS. Some items to keep in mind when defining this purpose include:

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. Defining this now will lead to less work in the future.

Product Scope

What are the benefits, objectives, and goals we intend to have for this product? This should relate to overall business goals, especially if teams outside of development will have access to the SRS.

Definitions and Acronyms

It's important to define the risks in the project. What could go wrong? How do me mitigate these risks? Who is in charge of these risk items?
For example, if the failure of a medical device would cause slight injury, that is one level of risk. Taking into account the occurrence level and the severity, we can come up with a strategy to mitigate this risk.

>> Need to create a PRD? Here's a how-to with examples >>

3. Describe What You Will Build

Your next step is to give a description of what you're going to build. Is it a new product? Is it an add-on to a product you've already created? Is this going to integrate with another product? Why is this needed? Who is it for?

Understanding these questions on the front end makes creating the product much easier for all involved.

User Needs

Describe who will use the product and how. Understanding the user of the product and their needs is a critical part of the process.

Who will be using the product? Are they a primary or secondary user? Do you need to know about the purchaser of the product as well as the end user? In medical devices, you will also need to know the needs of the patient.

Assumptions and Dependencies

What are we assuming will be true? Understating and laying out these assumptions ahead of time will help with headaches later. Are we assuming current technology? Are we basing this on a Windows framework? We need to take stock of these assumptions to better understand when our product would fail or not operate perfectly.

Finally, you should note if your project is dependent on any external factors. Are we reusing a bit of software from a previous project? This new project would then depend on that operating correctly and should be included.

4. Detail Your Specific Requirements

In order for your development team to meet the requirements properly, we MUST include as much detail as possible. This can feel overwhelming but becomes easier as you break down your requirements into categories. Some common categories are:

Functional Requirements

Functional requirements are essential to your product because, as they state, they provide some sort of functionality.

Asking yourself the question "does this add to my tool's functionality?" Or "What function does this provide?" can help with this process. Within Medical devices especially, these functional requirements may have a subset of risks and requirements.

You may also have requirements that outline how your software will interact with other tools, which brings us to external interface requirements.

External Interface Requirements

External interface requirements are specific types of functional requirements. These are especially important when working with embedded systems. 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

Nonfunctional requirements can be just as important as functional ones.

These include:

- Performance
- Safety
- Security
- Quality

5. Deliver for Approval

We made it! After completing the SRS, you'll need to get it approved by key stakeholders. This will require everyone to review the latest version of the document.

Behavioral Modeling Behavioral models describe the internal behavior of a system Behavioral model types: Representations of the details of a business process identified by use-cases Interaction diagrams (Sequence & Communication) Shows how objects collaborate to provide the functionality defined in the use cases. Representations of changes in the data Behavioral state machines like DFD

Interaction Diagrams Objects—an instantiation of a class Patient is a class Mary Wilson is an instantiation of the patient class (object) Attributes—characteristics of a class Patient class: name, address, phone, etc. Operations—the behaviors of a class, or an action that an object can perform Messages—information sent to objects to tell them to execute one of their behaviors A function call from one object to another Types Sequence Diagrams—emphasize message sequence Communication Diagrams—emphasize message flow

Data Dictionaries

A data dictionary is a file or a set of files that includes a database's metadata. The data dictionary hold records about other objects in the database, such as data ownership, data relationships to other objects, and other data. The data dictionary is an essential component of any relational database.

The data dictionary includes information about the following:

- Name of the data item
- Aliases
- Description/purpose
- Related data items
- Range of values
- Data structure definition/Forms

Software Quality framework

This is a framework that describes all the different concepts relating to quality in a common way measured by qualitative scale

1. **Developers View:**
   Validation and verification are two independent methods used together for checking that a software product meets the requirements and that it fulfills its intended purpose.Validation checks that the product design satisfies the purposeful usage and verification checks for errors in the software. The developer view of software quality and customer view of software quality are both different things.
   For example the customer understands or describes the quality of operation as meeting the requirement while the developers use different factors to describe the software quality.
   This model stresses on 3 primary ones:

   1. **The code:**
      It is measured by its correctness and reliability.
   2. **The data:**
      It is measured by the application integrity.
   3. **Maintainability:**
      It has different measures the simplest is the mean time to change.

2. **Users View:**
   When the user acquires software, he/she always expect a high-quality software. When end users develop their software then quality is different. End-user programming, a phrase popularized by which is programming to achieve the result of a program primarily for personal, rather than public use. The important distinction here is that software itself is not primarily intended for use by a large number of users with varying needs.

3. **Product View:**
   The product view describes quality as correlated to inherent characteristics of the product. Product quality is defined as the set of characteristics and features of a product that gives contribution to its ability to fulfill given requirements. Product quality can be measured by the value-based view which sees the quality as dependent on the amount a customer is willing to pay for it.

Software Quality Metrics for Analysis Model

There is a number of metrics available based on which software quality is measured. But among them, there are few most useful metrics which are most essential in software quality measurement. They are –

1. Code Quality
2. Reliability
3. Performance
4. Usability
5. Correctness
6. Maintainability
7. Integrity
8. Security

Now let's understand each quality metric **in detail –**

1. Code Quality – Code quality metrics measure the quality of code used for the software project development. Maintaining the software code quality by writing Bug-free and semantically correct code is very important for good software project development. In code quality both Quantitative

metrics like the number of lines, complexity, functions, rate of bugs generation, etc, and Qualitative metrics like readability, code clarity, efficiency, maintainability, etc are measured.
2. Reliability – Reliability metrics express the reliability of software in different conditions. The software is able to provide exact service at the right time or not is checked. Reliability can be checked using Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR).
3. Performance – Performance metrics are used to measure the performance of the software. Each software has been developed for some specific purposes. Performance metrics measure the performance of the software by determining whether the software is fulfilling the user requirements or not, by analyzing how much time and resource it is utilizing for providing the service.
4. Usability – Usability metrics check whether the program is user-friendly or not. Each software is used by the end-user. So it is important to measure that the end-user is happy or not by using this software.
5. Correctness – Correctness is one of the important software quality metrics as this checks whether the system or software is working correctly without any error by satisfying the user. Correctness gives the degree of service each function provides as per developed.
6. Maintainability – Each software product requires maintenance and up-gradation. Maintenance is an expensive and time-consuming process. So if the software product provides easy maintainability then we can say software quality is up to mark. Maintainability metrics include time requires to adapt to new features/functionality, Mean Time to Change (MTTC), performance in changing environments, etc.
7. Integrity – Software integrity is important in terms of how much it is easy to integrate with other required software's which increases software functionality and what is the control on integration from unauthorized software's which increases the chances of cyberattacks.
8. Security – Security metrics measure how much secure the software is? In the age of cyber terrorism, security is the most essential part of every software. Security assures that there are no unauthorized changes, no fear of cyber attacks, etc when the software product is in use by the end-user.

Software Metrics and Measures

Software Measures can be understood as a process of quantifying and symbolizing various attributes and aspects of software. Software Metrics provide measures for various aspects of software process and software product. Software measures are fundamental requirement of software engineering. They not only help to control the software development process but also aid to keep quality of ultimate product excellent. Let us see some software metrics:

- **Size Metrics** - LOC (Lines of Code), mostly calculated in thousands of delivered source code lines, denoted as KLOC. Function Point Count is measure of the functionality provided by the software. Function Point count defines the size of functional aspect of software.

- **Complexity Metrics** - McCabe's Cyclomatic complexity quantifies the upper bound of the number of independent paths in a program, which is perceived as complexity of the program or its modules. It is represented in terms of graph theory concepts by using control flow graph.

- **Quality Metrics** - Defects, their types and causes, consequence, intensity of severity and their implications define the quality of product. The number of defects found in development process and number of defects reported by the client after the product is installed or delivered at client-end, define quality of product.

- **Process Metrics** - In various phases of SDLC, the methods and tools used, the company standards and the performance of development are software process metrics.

- **Resource Metrics** - Effort, time and various resources used, represents metrics for resource measurement.

Software Process Framework

- The process of framework defines a small set of activities that are applicable to all types of projects.
- The software process framework is a collection of task sets.
- Task sets consist of a collection of small work tasks, project milestones, work productivity and software quality assurance points.

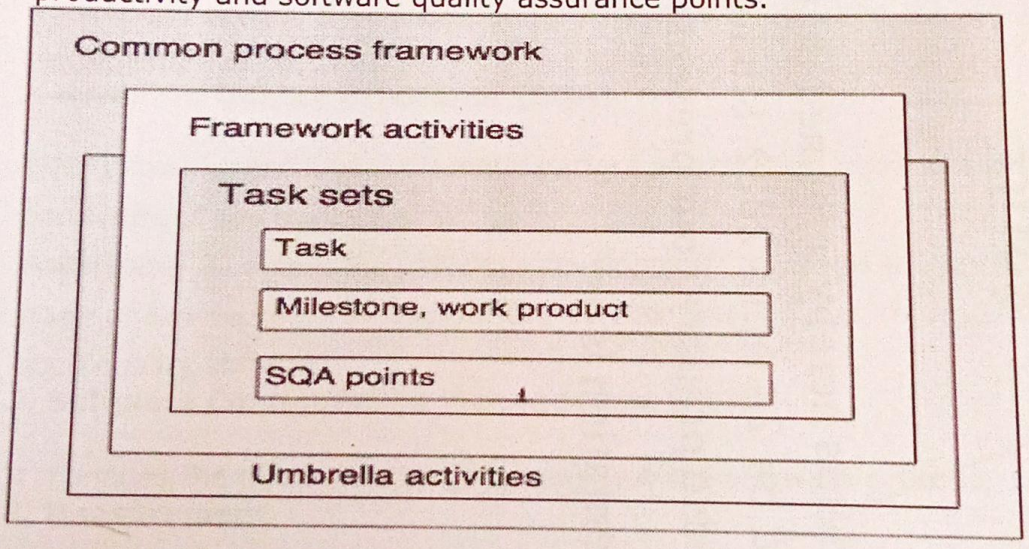


Fig.- A software process framework

A generic process framework encompasses five activities which are given below one by one:

1. **Communication:**
   In this activity, heavy communication with customers and other stakeholders, as well as requirement gathering is done.
2. **Planning:**
   In this activity, we discuss the technical related tasks, work schedule, risks, required resources, etc.
3. **Modeling:**
   Modeling is about building representations of things in the 'real world'. In modeling activity, a product's model is created in order to better understand the requirements.
4. **Construction:**
   In software engineering, construction is the application of set of procedures that are needed to

assemble the product. In this activity, we generate the code and test the product in order to make better product.

5. **Deployment:**
   In this activity, a complete or non-complete product or software is represented to the customers to evaluate and give feedback. On the basis of their feedback, we modify the product for supply of better product.

Umbrella activities include:

- Risk Management
- Software Quality Assurance (SQA)
- Software Configuration Management (SCM)
- Measurement
- Formal Technical Reviews (FTR)

1. Risk management

- Risk is an event that may or may not occur.
- If the event occurs, then it causes some unwanted outcome. Hence, proper risk management is required.

2. Software Quality Assurance (SQA)

- SQA is the planned and systematic pattern of activities which are required to give a guarantee of software quality.

  For example, during the software development meetings are conducted at every stage of development to find out the defects and suggest improvements to produce good quality software.

3. Software Configuration Management (SCM)

It manages the effect of change throughout the software process.

4. Measurement

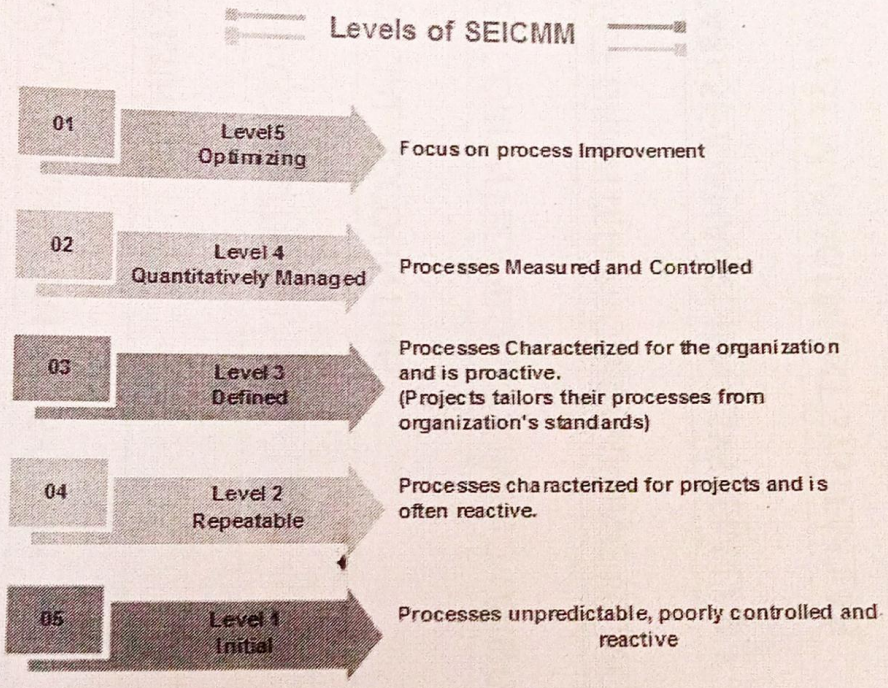
- Measurement consists of the effort required to measure the software.
- The software cannot be measured directly. It is measured by direct and indirect measures.
- Direct measures like cost, lines of code, size of software etc.
- Indirect measures such as quality of software which is measured by some other factor. Hence, it is an indirect measure of software.

5. Formal Technical Reviews (FTR)

- FTR is a meeting conducted by the technical staff.
- The motive of the meeting is to detect quality problems and suggest improvements.

- The technical person focuses on the quality of the software from the customer point of view.

CMM establishes a framework for continuous process improvement. It is more explicit than the ISO standard in defining the means to be employed to that end.



CMM's five levels of maturity for software processes

There are five levels to the CMM development process. They are the following:

1. **Initial.** At the initial level, processes are disorganized, ad hoc and even chaotic. Success likely depends on individual efforts and is not considered to be repeatable. This is because processes are not sufficiently defined and documented to enable them to be replicated.
2. **Repeatable.** At the repeatable level, requisite processes are established, defined and documented. As a result, basic project management techniques are established, and successes in key process areas are able to be repeated.