KOMMURI PRATAP REDDY INSTITUTE OF TECHNOLOGY

Academic Year: 2019-2020

Name of the Course: Software Engineering

Course Code: CS502PC

Year and Semester: III/I

Name of the Faculty: Mr. A. Sai Kumar

Department in which subject is handled: CSE

Course Type: Basic Sciences / Humanities & Social Sciences/ Professional Core / Professional elective / Open Elective /Engineering Sciences / Mandatory courses / Project.

Vision of the Institute

To emerge as a premier institute for high quality professional graduates who can contribute to economic and social developments of the Nation.

Mission of the Institute

Mission	Statement
IM1	To have holistic approach in curriculum and pedagogy through industry interface to meet the needs of Global Competency.
IM2	To develop students with knowledge, attitude, employability skills, entrepreneurship, research potential and professionally ethical citizens.
IM3	To contribute to advancement of Engineering & Technology that would help to satisfy the societal needs.
IM4	To preserve, promote cultural heritage, humanistic values and spiritual values thus helping in peace and harmony in the society.

Vision of the Department

To Provide Quality Education in Computer Science for the innovative professionals to work for the development of the nation.

Mission of the Department

Mission	Statement
DM1	Laying the path for rich skills in Computer Science through the basic knowledge of mathematics and fundamentals of engineering
DM2	Provide latest tools and technology to the students as a part of learning infrastructure
DM3	Training the students towards employability and entrepreneurship to meet the societal needs.
DM4	Grooming the students with professional and social ethics.

Program Educational Objectives:

PEO1: The graduates of Computer Science and Engineering will have successful career in technology.

PEO2: The graduates of the program will have solid technical and professional foundation to continue higher studies.

PEO3: The graduate of the program will have skills to develop products, offer services and innovation.

PEO4: The graduates of the program will have fundamental awareness of industry process, tools and technologies.

Program Outcomes (POs)

Engineering Graduates will be able to:

PO1: Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. Problem Analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental context, and demonstrate the knowledge of, and need for sustainable development.

PO8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. Individual and team network: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12. Life-Long learning: Recognize the need for, and have the preparation and able to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

PSO1. Foundation of mathematical concepts: To use mathematical methodologies to crack problem using suitable mathematical analysis, data structure and suitable algorithm.

PSO2. Foundation of Computer Science: The ability to interpret the fundamental concepts and methodology of computer systems. Students can understand the functionality of hardware and software aspects of computer systems.

PSO3. Foundation of Software development: The ability to grasp the software development lifecycle and methodologies of software systems. Possess competent skills and knowledge of software design process.

KOMMURI PRATAP REDDY INSTITUTE OF TECHNOLOGY

Name of the Faculty: A. Sai Kumar **Academic Year: 2021-22**

Subject: Software Engineering **Year: III / Semester: I**

Course Outcomes

Course Name: Software Engineering (C312)

C312.1: Define the evolving role and changing nature of the software and also the generic view of process and process models. [Remember]

C312.2:Explainsabout the software requirements, requirement engineering process and process models.[Understand]

C312.3:Implements to translate end-user requirements into system and software requirements, using e.g. UML, and structure the requirements in a Software Requirements Document (SRD).[Apply]

C312.4:Examine the appropriate software architectures and patterns to carry out high level design of a system and be able to critically compare alternative choices.[Analyze]

C312.5:Investigate about the software metrics and RMMM, and also they will have awareness on the quality management of the software.[Create]

Faculty

KOMMURI PRATAP REDDY INSTITUTE OF TECHNOLOGY

CO- PO& PSO Mapping

Course Name: Software Engineering (C312)

PO / CO	PO1	PO2	PO3	PO 4	PO5	PO 6	PO 7	PO8	PO 9	PO 10	PO 11	PO1 2	PSO1	PSO 2	PSO 3
C312.1	3	2	1	2	3	-	-	2	2	2	2	3	-	2	3
C312.2	-	3	2	1	3	-	-	2	3	2	2	-	-	2	3
C312.3	-	2	3	2	2	2	-	-	2	2	2	1	-	2	3
C312.4	3	2	3	2	3	-	1	2	2	3	1	-	-	3	2
C312.5	3	3	2	3	3	-	-	2	3	3	2	1	2	2	3
C312	1.8	2.4	2.2	2	2.8	0.4	0.2	1.6	2.4	2.4	1.8	1	0.4	2.2	2.8

Level of Mapping: High -3, Medium -2, Low-1

Faculty

CO-PO mapping Justification

C312.1: Define the evolving role and changing nature of the software and also the generic view of process and process models. [Remember]

	Justification
PO1	Students will be able to know the concept of software engineering.
PO2	Students will be able to define the evolving role of software engineering and the changing nature of software and its applications.
PO3	Students will be able to define the generic view of the process and its framework.
PO4	Students will be able to define the process patterns and process assessment.
PO5	Students will be able to state the software development life cycle.
PO8	Students will be able to define the different SDLC models.
PO9	Students will be able memorizes the different SDLC models.
PO10	Students will be able to communicate about the process models.
PO11	Students will be able to know the how to design a process framework.
PO12	Students will be able to remember the all the concepts of the Generic view of process and process models

C312.2: Explains about the software requirements, requirement engineering process and process models.[Understand]

	Justification
PO2	Students will able to identify about the different requirements of the requirement engineering.
PO3	Students will able to design the different requirements into the SRS.
PO4	Students will have the knowledge about the user, system, functional and non - functional requirements.
PO5	Students will be describing the different tools and techniques for requirement elicitation and analysis, requirement management.
PO8	Students will be able understands the principles of the requirement engineering.
PO9	Students will be able to classifies the different types of system models
PO10	Students will be able to discuss about the context,data,object,behavioural models.
PO11	Students will be able to describes the requirements of the projects.

C312.3: Implements to translate end-user requirements into system and software requirements, using e.g. UML, and structure the requirements in a Software Requirements Document (SRD).[Apply]

	Justification
PO2	Students will be able to implement the end- user requirements.
PO3	Students will be able to design the process design, quality, concepts.
PO4	Students will be able uses different design concepts for the investigation of the architecture.
PO5	Students will be able uses different techniques to design the system models.
PO6	Students will be able solves the engineering practices for the design of the styles and patterns.
PO9	Students will be able to schedule about the design models using UML.

PO10	Students will be able to demonstrate the different UML Diagrams.
PO11	Students will be able to execute the class diagrams, use case diagram, sequence diagram etc.
PO12	Students will be ableto implements the different design concepts using the UML Diagrams.

C312.4: Examine the appropriate software architectures and patterns to carry out high level design of a system and be able to critically compare alternative choices.[Analyze]

	Justification
PO1	Students will be able to have the knowledge about the architecture for the design model.
PO2	Students will be able to analyze about different types of architecture.
PO3	Students will be able to develop the architecture of the system models.
PO4	Students will be able to examine the architecture styles and patterns.
PO5	Students will be able to distinguish different engineering methods and techniques to design the architecture of the system model.
PO7	Students will be ableto examine the appropriate software architecture.
PO8	Students will be able to analyze the principles of the software architecture.
PO9	Students will be able to organize the work to design the software architecture.
PO10	Students will be able to communicate the different styles and patterens of the architecture.
PO11	Students will be able to analyses the development of the project.

C312.5: Investigate about the software metrics and RMMM, and also they will have awareness on the quality management of the software. [Create]

	Justification
PO2	Students will be able to identify the software metrics
PO3	Students will be able to develop the quality metrics.
PO4	Students will be able to investigate the RMMM.
PO5	Students will be able to use the modern techniques to find out the risk management in the project.
PO7	Students will be able to explain about the RMMM plan to develop the project.
PO9	Students will be able to explain about the concept of quality management.
PO10	Students will be able to explain about the ISO 9000 quality standards.
PO11	Students will be able to develop the quality software project.
PO12	Students will be able to understands how create/develop a software project with quality 6to the client.

CO-PSO mapping Justification

C312.1: Define the evolving role and changing nature of the software and also the generic view of process and process models. [Remember]

	Justification
PSO2	Students can understand the changing nature of the software
PSO3	Students can able to grasp the process models

C312.2: Explains about the software requirements, requirement engineering process and process models. [Understand]

	Justification
PSO2	Students are able to apply the fundamental concepts and software requirements and requirement engineering
PSO3	Student get the knowledge of process models in software design process

C312.3: Implements to translate end-user requirements into system and software requirements, using e.g. UML, and structure the requirements in a Software Requirements Document (SRD). [Apply]

	Justification
PSO2	Get the basic knowledge on end user requirements and designing the uml diagrams
PSO3	Students can get the knowledge in preparation Software Require Document (SRD)

C312.4: Examine the appropriate software architectures and patterns to carry out high level design of a system and be able to critically compare alternative choices.[Analyze]

	Justification
PSO2	Students get the knowledge on preparing software architectures and patterns
PSO3	Students able to apply the knowledge to preparing high level design of a system and able to compare alternative choices

C312.5: Investigate about the software metrics and RMMM, and also they will have awareness on the quality management of the software. [Create]

	Justification
PSO1	Students can able to crack the problems of software metrics
PSO2	Students are able to having understand the software quality management
PSO3	Students can able to creating knowledge on the good quality of software

CS502PC: SOFTWARE ENGINEERING

III Year B.Tech. CSE I-Sem

L	T	P	C
3	0	0	3

Course Objectives

1. The aim of the course is to provide an understanding of the working knowledge of the techniques for estimation, design, testing and quality management of large software development projects.
2. Topics include process models, software requirements, software design, software testing, software process/product metrics, risk management, quality management and UML diagrams

Course Outcomes

1. Ability to translate end-user requirements into system and software requirements, using e.g.UML, and structure the requirements in a Software Requirements Document (SRD).
2. Identify and apply appropriate software architectures and patterns to carry out high level designof a system and be able to critically compare alternative choices.
3. Will have experience and/or awareness of testing problems and will be able to develop a simpletesting report

UNIT - I

Introduction to Software Engineering: The evolving role of software, changing nature of software, software myths.

A Generic view of process: Software engineering- a layered technology, a process framework, the capability maturity model integration (CMMI), process patterns, process assessment, personal and team process models.

Process models: The waterfall model, incremental process models, evolutionary process models, the unified process.

UNIT - II

Software Requirements: Functional and non-functional requirements, user requirements, system requirements, interface specification, the software requirements document.

Requirements engineering process: Feasibility studies, requirements elicitation and analysis, requirements validation, requirements management.

System models: Context models, behavioral models, data models, object models, structured methods.

UNIT - III

Design Engineering: Design process and design quality, design concepts, the design model.
Creating an architectural design: software architecture, data design, architectural styles and patterns, architectural design, conceptual model of UML, basic structural modeling, class diagrams, sequence diagrams, collaboration diagrams, use case diagrams, component diagrams.

UNIT - IV

Testing Strategies: A strategic approach to software testing, test strategies for conventional software, black-box and white-box testing, validation testing, system testing, the art of debugging.
Product metrics: Software quality, metrics for analysis model, metrics for design model, metrics for source code, metrics for testing, metrics for maintenance.

UNIT - V

Metrics for Process and Products: Software measurement, metrics for software quality.

Risk management: Reactive Vs proactive risk strategies, software risks, risk identification, risk projection, risk refinement, RMMM, RMMM plan.

Quality Management: Quality concepts, software quality assurance, software reviews, formal technical reviews, statistical software quality assurance, software reliability, the ISO 9000 quality standards.

TEXT BOOKS:

1. Software Engineering, A practitioner's Approach- Roger S. Pressman, 6th edition, Mc Graw Hill International Edition.
2. Software Engineering- Sommerville, 7th edition, Pearson Education.
3. The unified modeling language user guide Grady Booch, James Rambaugh, Ivar Jacobson,Pearson Education.

REFERENCES:

1. Software Engineering, an Engineering approach- James F. Peters, Witold Pedrycz, John Wiley.
2. Software Engineering principles and practice- Waman S Jawadekar, The Mc Graw-HillCompanies.
3. Fundamentals of object-oriented design using UML Meiler page-Jones: Pearson Education.

Unit-1

Introduction to Software Engineering

The term is made of two words, software and engineering.

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

Software is defined as

1. Instructions

Programs that when executed provide desired function

2. Data structures

Enable the programs to adequately manipulate information

3. Documents

Describe the operation and use of the programs.

Definition of Engineering

Application of science, tools and methods to find cost effective solution to problems.

Definition of Software Engineering

SE is defined as systematic, disciplined and quantifiable approach for the development, operation and maintenance of software.

Software Engineering | Software Evolution

Software Evolution is a term which refers to the process of developing software initially, then timely updating it for various reasons, i.e., to add new features or to remove obsolete functionalities etc. The evolution process includes fundamental activities of change analysis, release planning, system implementation and releasing a system to customers.

The cost and impact of these changes are accessed to see how much system is affected by the change and how much it might cost to implement the change. If the proposed changes are accepted, a new release of the software system is planned. During release planning, all the proposed changes (fault repair, adaptation, and new functionality) are considered.

A design is then made on which changes to implement in the next version of the system. The process of change implementation is an iteration of the development process where the revisions to the system are designed, implemented and tested.

The necessity of Software evolution: Software evaluation is necessary just because of the following reasons:

a) Change in requirement with time: With the passes of time, the organization's needs and modus Operandi of working could substantially be changed so in this frequently changing time the tools (software) that they are using need to change for maximizing the performance.

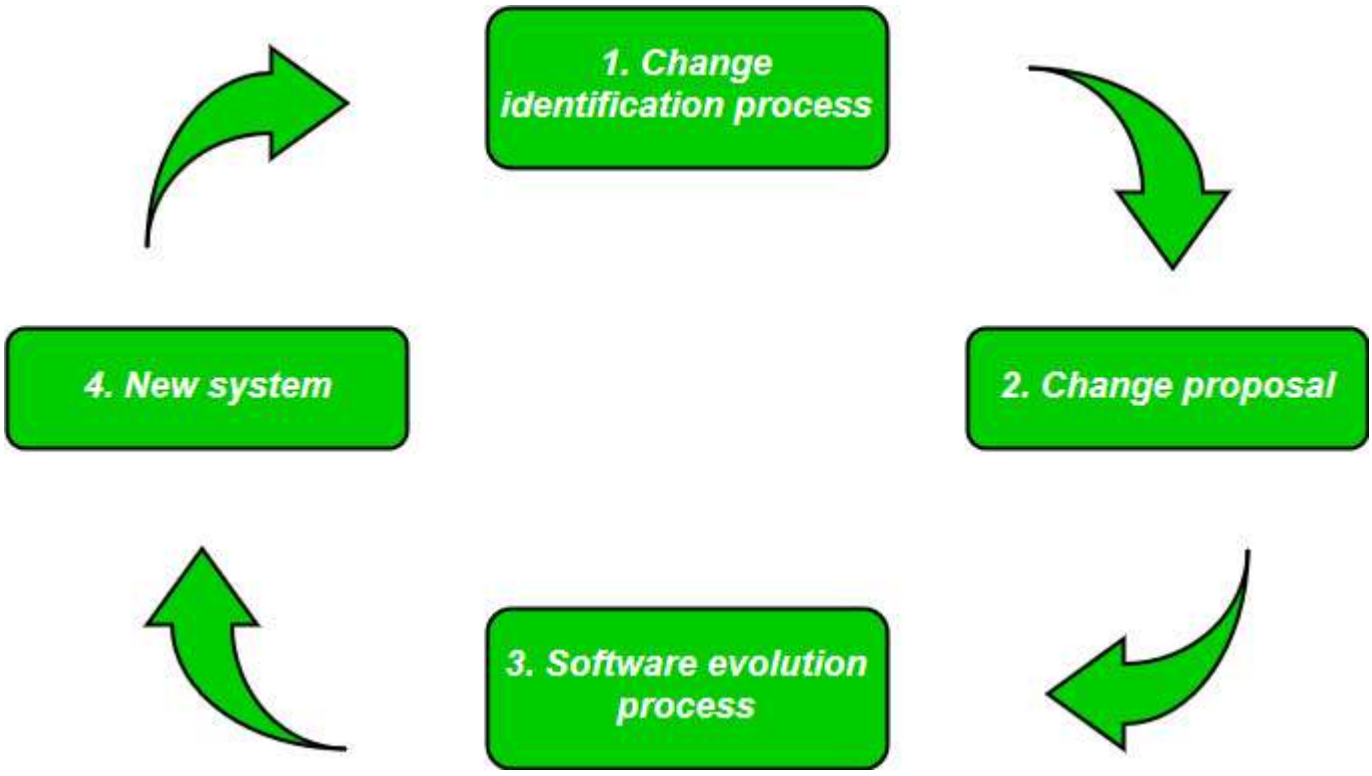
b) Environment change: As the working environment changes the things(tools) that enable us to work in that environment also changes proportionally same happens in the software world as the working environment changes then, the organizations need reintroduction of old software with updated features and functionality to adapt the new environment.

c) Errors and bugs: As the age of the deployed software within an organization increases their preciseness or impeccability decrease and the efficiency to bear the increasing complexity workload also continually degrades. So, in that case, it becomes necessary to avoid use of obsolete and aged software. All such obsolete Softwares need to undergo the evolution process in order to become robust as per the workload complexity of the current environment.

d) Security risks: Using outdated software within an organization may lead you to at the verge of various software-based cyber attacks and could expose your confidential data illegally associated with the software that is in use. So, it becomes necessary to avoid such security breaches through regular assessment of the security patches/modules are used within the

software. If the software isn't robust enough to bear the current occurring Cyber attacks so it must be changed (updated).

e) <u>For having new functionality and features</u>: In order to increase the performance and fast data processing and other functionalities, an organization need to continuously evolute the software throughout its life cycle so that stakeholders & clients of the product could work efficiently.



Laws used for Software Evolution:

1. **Law of continuing change:**
   This law states that any software system that represents some real-world reality undergoes continuous change or become progressively less useful in that environment.
2. **Law of increasing complexity:**
   As an evolving program changes, its structure becomes more complex unless effective efforts are made to avoid this phenomenon.

3. **Law of conservation of organization stability:**
   Over the lifetime of a program, the rate of development of that program is approximately constant and independent of the resource devoted to system development.
4. **Law of conservation of familiarity:**
   This law states that during the active lifetime of the program, changes made in the successive release are almost constant.

Characteristics of software

- **Software is developed or engineered** it is not manufactured in the classical sense.

- **Software does not wear out. However**it deteriorates due to change.

- **Software is custom built** rather than assembling existing components.

Although the industry is moving towards component based construction, most software continues to be custom built.

Changing Nature of Software

Nowadays, seven broad categories of computer software present continuing challenges for software engineers .which is given below:

1. **System Software:**
   System software is a collection of programs which are written to service other programs. Some system software processes complex but determinate, information structures. Other system application process largely indeterminate data. Sometimes when, the system software area is characterized by the heavy interaction with computer hardware that requires scheduling, resource sharing, and sophisticated process management.
   Ex: - Examples of system software include operating systems like macOS, GNU/Linux , Android and Microsoft Windows, computational science software, game engines, industrial automation, and software as a service applications.

2. **Application Software:**

   Application software is defined as programs that solve a specific business need. Application in this area process business or technical data in a way that facilitates business operation or management technical decision making. In addition to convention data processing application, application software is used to control business function in real time.

   Ex:-

- Microsoft suite of products (Office, Excel, Word, PowerPoint, Outlook, etc.)
- Internet browsers like Firefox, Safari, and Chrome.
- Mobile pieces of **software** such as Pandora (for music appreciation), Skype (for real-time online communication), and Slack (for team collaboration)
- **Application software** (app for short) is a program or group of programs designed for end users. **Examples** of an **application** include a word processor, a spreadsheet, an accounting **application**, a web browser, an email client, a media player, a file viewer, simulators, a console game or a photo editor.

3. **Engineering and Scientific Software:**

   This software is used to facilitate the engineering function and task. However modern application within the engineering and scientific area are moving away from the conventional numerical algorithms. Computer-aided design, system simulation, and other interactive applications have begun to take a real-time and even system software characteristic.

   Ex: - Examples are **software** like MATLAB, AUTOCAD, PSPICE, ORCAD, etc.

4. **Embedded Software:**

   Embedded software resides within the system or product and is used to implement and control feature and function for the end-user and for the system itself. Embedded software can perform the limited and esoteric function or provided significant function and control capability.

This type of software is embedded into the hardware normally in the Read Only Memory (ROM) as a part of a large system and is used to support certain functionality under the control conditions.

Examples are software used in instrumentation and control applications, washing machines, satellites, microwaves, washing machines etc.

5. **Product-line Software:**

Designed to provide a specific capability for use by many different customers, product line software can focus on the limited and esoteric marketplace or address the mass consumer market.

6. **Web Application:**

It is a client-server computer program which the client runs on the web browser. In their simplest form, Web apps can be little more than a set of linked hypertext files that present information using text and limited graphics. However, as e-commerce and B2B application grow in importance. Web apps are evolving into a sophisticate computing environment that not only provides a standalone feature, computing function, and content to the end user.

Web applications include online forms, shopping carts, word processors, spreadsheets, video and photo editing, file conversion, file scanning, and email programs such as Gmail, Yahoo and AOL. Popular **applications** include Google **Apps** and Microsoft 365.Online retail sales, online auctions, wikis, instant messaging services and more.

7. **Artificial Intelligence Software:**

Artificial intelligence software makes use of a nonnumerical algorithm to solve a complex problem that is not amenable to computation or straightforward analysis. Application within this area includes robotics, expert system, pattern recognition, artificial neural network, theorem proving and game playing.

Example, speech recognition, problem-solving, learning and planning.

Artificial Intelligence Examples

- Manufacturing robots.
- Smart assistants.
- Proactive healthcare management.
- Disease mapping.
- Automated financial investing.
- Virtual travel booking agent.
- Social media monitoring.
- Inter-team chat tool.

Software Development Myths

Pressman (1997) describes a number of common beliefs or myths that software managers, customers, and developers believe falsely. He describes these myths as ``misleading attitudes that have caused serious problems." We look at these myths to see why they are false, and why they lead to trouble.

Software Management Myths: - Pressman describes managers' beliefs in the following mythology as grasping at straws:

- *Development problems can be solved by developing and documenting standards.* Standards have been developed by companies and standards organizations. They can be very useful. However, they are frequently ignored by developers because they are irrelevant and incomplete, and sometimes incomprehensible.
- *Development problems can be solved by using state-of-the art tools.* Tools may help, but there is no magic. Problem solving requires more than tools, it requires great understanding. As Fred Brooks (1987) says, there is no silver bullet to slay the software development werewolf.
- *When schedules slip, just add more people* This solution seems intuitive: if there is too much work for the current team, just enlarge it. Unfortunately, increasing team size increases communication overhead. New workers must learn project details taking up the

time of those who are already immersed in the project. Also, a larger team has many more communication links, which slows progress. Fred Brooks (1975) gives us one of the most famous software engineering maxims, **which is not a myth**, ``adding people to a late project makes it later."

Software Customer Myths: - Customers often vastly underestimate the difficulty of developing software. Sometimes marketing people encourage customers in their misbelieves.

- *Change is easily accommodated, since software is malleable.*

Software can certainly be changed, but often changes after release can require an enormous amount of labor.

- *A general statement of need is sufficient to start coding*

This myth reminds me of a cartoon that I used to post on my door. It showed the software manager talking to a group of programmers, with the quote: ``You programmers just start coding while I go down and find out what they want the program to do." This scenario is an exaggeration. However, for developers to have a chance to satisfy the customers requirements, they need detailed descriptions of these requirements. Developers cannot read the minds of customers.

Developer Myths: - Developers often want to be artists (or artisans), but the software development craft is becoming an engineering discipline. However myths remain:

- *The job is done when the code is delivered.*

Commercially successful software may be used for decades. Developers must continually maintain such software: they add features and repair bugs. Maintenance costs predominate over all other costs; maintenance may be 70% of the development costs. This myth is true only for *shelfware* --- software that is never used, and there are no customers for next release of a shelfware product.

- *Project success depends solely on the quality of the delivered **program***.

Documentation and software configuration information is very important to the quality. After functionality, maintainability, see the preceding myth, is of critical importance. Developers must maintain the software and they need good design documents, test data, etc to do their job.

- *You can't assess software quality until the program is running.*

There are *static* ways to evaluate quality without running a program. Software reviews can effectively determine the quality of requirements documents, design documents, test plans, and code. Formal (mathematical) analyses are often used to verify safety critical software, software security factors, and very-high reliability software.

Software Engineering - Layered Technology

- Software development is totally a layered technology.
- To develop software, we need to go from one layer to another.
- All these layers are related to each other and each layer demands the fulfillment of the previous layer.

Layers of Software Development

1. Quality Focus
2. Process
3. Methods
4. Tools

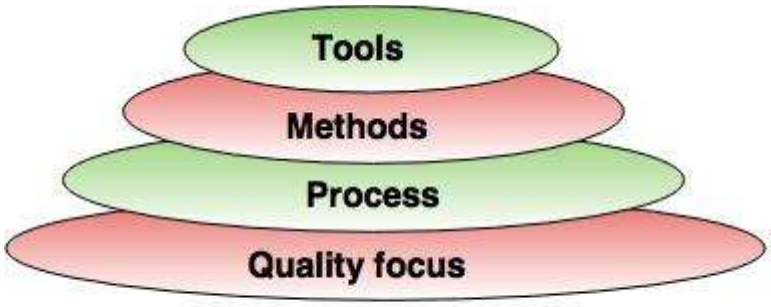


Fig. - Software Engineering Layers

1. Quality focus

The characteristics of good quality software are:

- Software engineering must rest on an organizational commitment to quality.
- The "Bed Rock" that supports software Engineering is Quality Focus.

- Correctness of the functions required to be performed by the software.
- Maintainability of the software
- Integrity i.e. providing security so that the unauthorized user cannot access information or data.
- Usability i.e. the efforts required to use or operate the software.

2. Process

- It is the base layer or foundation layer for the software engineering.
- The software process is the key to keep all levels together.
- Process defines a framework for a set of Key Process Areas (KPAs) that must be established for effective delivery of software engineering technology. Consequently, this establishes the context in which technical methods are applied, work products such as models, documents, data, reports, forms, etc. are produced, milestones are established, quality is ensured, and change is properly managed.
- It defines a framework that includes different activities and tasks.
- In short, it covers all activities, actions and tasks required to be carried out for software development.

3. Methods

- The method provides the answers of all 'how-to' that are asked during the process.

- Software engineering methods provide the technical how-to's for building software.
- It provides the technical way to implement the software.
- It includes collection of tasks starting from communication, requirement analysis, analysis and design modelling, program construction, testing and support.
- This relies on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

4. Tools

- Software engineering tools provide automated or semi-automated support for the process and the methods.
- When tools are integrated so that information created by one tool can be used by another.
- A system for the support of software development, called computer-aided software engineering, is established.

For example: The Microsoft publisher can be used as a web designing tool.

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.

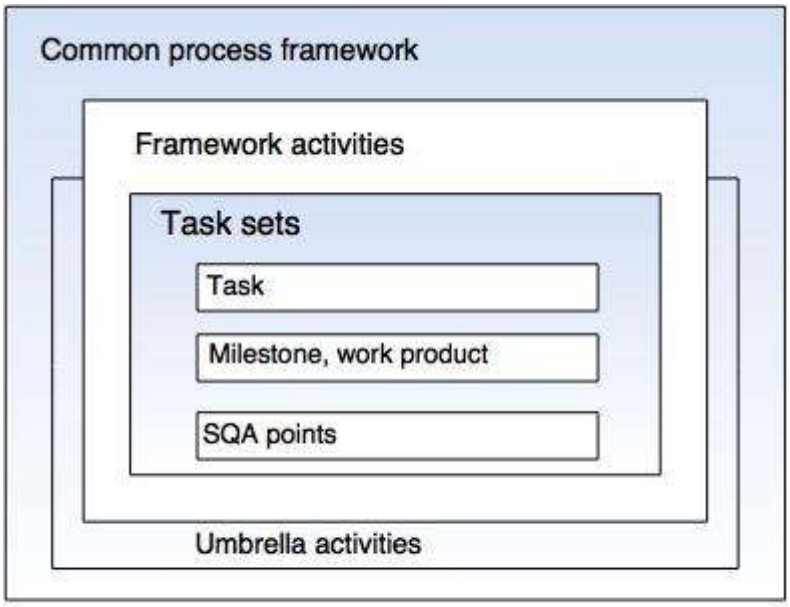


Fig.- A software process framework

Umbrella activities

Typical umbrella activities are:

1. Software project tracking and control

- ✓ In this activity, the developing team accesses project plan and compares it with the predefined schedule.
- ✓ If these project plans do not match with the predefined schedule, then the required actions are taken to maintain the schedule.

2. 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.

3. 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.

4. 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.

5. 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.

6. Software Configuration Management (SCM)

- ✓ It manages the effect of change throughout the software process.

7. Reusability management

- ✓ It defines the criteria for reuse the product.
- ✓ The quality of software is good when the components of the software are developed for certain application and are useful for developing other applications.

8. Work product preparation and production

- ✓ It consists of the activities that are needed to create the documents, forms, lists, logs and user manuals for developing software.

Generic Process Model

A software process is a collection of various activities.

There are five generic process framework activities:

1. Communication:

The software development starts with the communication between customer and developer.

2. Planning:

It consists of complete estimation, scheduling for project development and tracking.

3. Modeling:

- ✓ Modeling consists of complete requirement analysis and the design of the project like algorithm, flowchart etc.
- ✓ The algorithm is the step-by-step solution of the problem and the flow chart shows a complete flow diagram of a program.

4. Construction:

- ✓ Construction consists of code generation and the testing part.
- ✓ Coding part implements the design details using an appropriate programming language.
- ✓ Testing is to check whether the flow of coding is correct or not.
- ✓ Testing also check that the program provides desired output.

5. Deployment:

- ✓ Deployment step consists of delivering the product to the customer and take feedback from them.
- ✓ If the customer wants some corrections or demands for the additional capabilities, then the change is required for improvement in the quality of the software.

The Capability Maturity Model Integration (CMMI)

What is CMMI?

The Capability Maturity Model Integration (CMMI) is a capability maturity model developed by the Software Engineering Institute, part of Carnegie Mellon University in Pittsburgh, USA. The CMMI principal is that "the quality of a system or product is highly influenced by the process used to develop and maintain it". CMMI can be used to guide process improvement across a project, a division, or an entire organization.

CMMI provides:

- Guidelines for processes improvement
- An integrated approach to process improvement
- Embedding process improvements into a state of business as usual
- A phased approach to introducing improvements

CMMI Models

CMMI consists of three overlapping disciplines (constellations) providing specific focus into the Development, Acquisition and Service Management domains respectively:

- CMMI for Development (CMMI-DEV) – Product and service development
- CMMI for Services (CMMI-SVC) – Service establishment, management, and delivery

- CMMI for Acquisition (CMMI-ACQ) – Product and service acquisition

Originating in software engineering, CMMI has been highly generalized over the years to embrace other business processes such as the development of hardware products, service delivery and purchasing which has had the effect of abstracting CMMI.

CMMI Model Framework

Depending on the CMMI constellation (CMMI-DEV, CMMI-SVC & CMMI ACQ) used, the process areas it contains will vary. The table below lists the process areas that are present in all CMMI constellations. This collection of eight process areas is called the CMMI Model Framework, or CMF.

Capability Maturity Model Integration (CMMI) Model Framework (CMF)			
Abbreviation	**Name**	**Area**	**Maturity Level**
REQM	Requirements Management	Engineering	2
PMC	Project Monitoring and Control	Project Management	2
PP	Project Planning	Project Management	2
CM	Configuration Management	Support	2
MA	Measurement and Analysis	Support	2
PPQA	Process and Product Quality Assurance	Support	2
OPD	Organizational Process Definition	Process Management	3
CAR	Causal Analysis	Support	5

CMMI Maturity Levels

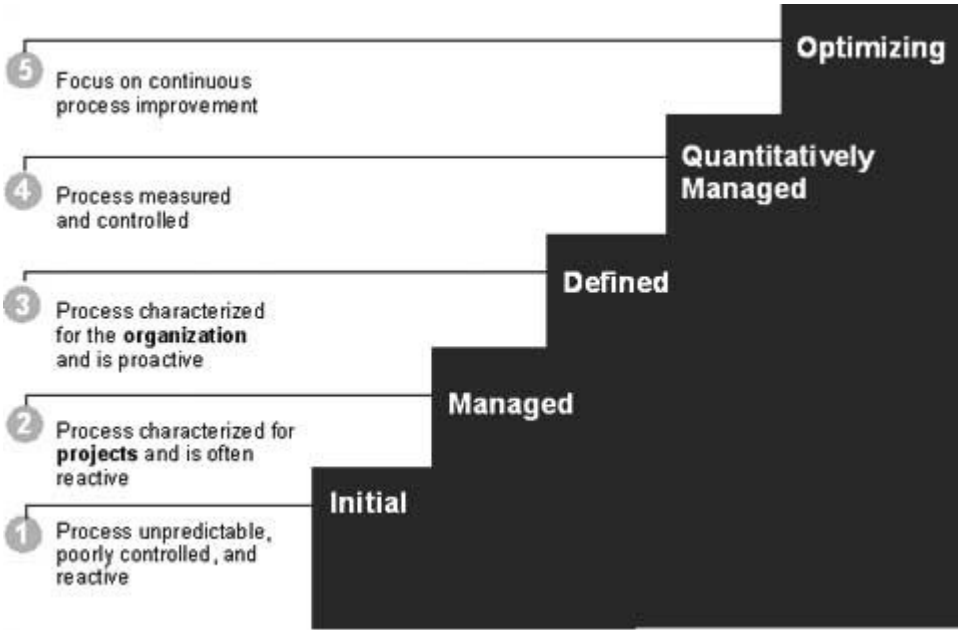
A maturity level is a well-defined evolutionary plateau toward achieving a mature software process. Each maturity level provides a layer in the foundation for continuous process improvement.

In CMMI models with a staged representation, there are five maturity levels designated by the numbers 1 through 5

There are five CMMI maturity levels. However, maturity level ratings are only awarded for levels 2 through 5.

1. Initial
2. Managed
3. Defined
4. Quantitatively Managed
5. Optimizing

CMMI Staged Representation- Maturity Levels



Now we will give more detail about each maturity level. Next section will list down all the process areas related to these maturity levels.

Maturity Level Details:

Maturity levels consist of a predefined set of process areas. The maturity levels are measured by the achievement of the **specific** and **generic goals** that apply to each predefined set of process areas. The following sections describe the characteristics of each maturity level in detail.

Maturity Level 1 - Initial

At maturity level 1, processes are usually ad hoc and chaotic. The organization usually does not provide a stable environment. Success in these organizations depends on the competence and heroics of the people in the organization and not on the use of proven processes.

Maturity level 1 organizations often produce products and services that work; however, they frequently exceed the budget and schedule of their projects.

Maturity level 1 organizations are characterized by a tendency to over commit, abandon processes in the time of crisis, and not be able to repeat their past successes.

CMMI Maturity Level 2 – Managed

- CM – Configuration Management
- MA – Measurement and Analysis
- PMC – Project Monitoring and Control
- PP – Project Planning
- PPQA – Process and Product Quality Assurance
- REQM – Requirements Management
- SAM – Supplier Agreement Management

At maturity level 2, an organization has achieved all the **specific** and **generic goals** of the maturity level 2 process areas. In other words, the projects of the organization have ensured that requirements are managed and that processes are planned, performed, measured, and controlled.

The process discipline reflected by maturity level 2 helps to ensure that existing practices are retained during times of stress. When these practices are in place, projects are performed and managed according to their documented plans.

At maturity level 2, requirements, processes, work products, and services are managed. The status of the work products and the delivery of services are visible to management at defined points.

Commitments are established among relevant stakeholders and are revised as needed. Work products are reviewed with stakeholders and are controlled.

The work products and services satisfy their specified requirements, standards, and objectives.

CMMI Maturity Level 3 – Defined

- DAR – Decision Analysis and Resolution
- IPM – Integrated Project Management +IPPD
- OPD – Organizational Process Definition +IPPD
- OPF – Organizational Process Focus
- OT – Organizational Training
- PI – Product Integration
- RD – Requirements Development
- RSKM – Risk Management
- TS – Technical Solution
- VAL – Validation
- VER – Verification

At maturity level 3, an organization has achieved all the **specific** and **generic goals** of the process areas assigned to maturity levels 2 and 3.

At maturity level 3, processes are well characterized and understood, and are described in standards, procedures, tools, and methods.

A critical distinction between maturity level 2 and maturity level 3 is the scope of standards, process descriptions, and procedures. At maturity level 2, the standards, process descriptions, and procedures may be quite different in each specific instance of the process (for example, on a particular project). At maturity level 3, the standards, process descriptions, and procedures for a project are tailored from the organization's set of standard processes to suit a particular project or organizational unit. The organization's set of standard processes includes the processes addressed at maturity level 2 and maturity level 3. As a result, the processes that are performed across the organization are consistent except for the differences allowed by the tailoring guidelines.

Another critical distinction is that at maturity level 3, processes are typically described in more detail and more rigorously than at maturity level 2. At maturity level 3, processes are managed more proactively using an understanding of the interrelationships of the process activities and detailed measures of the process, its work products, and its services.

CMMI Maturity Level 4 – Quantitatively Managed

- QPM – Quantitative Project Management
- OPP – Organizational Process Performance

At maturity level 4, an organization has achieved all the **specific goals** of the process areas assigned to maturity levels 2, 3, and 4 and the **generic goals** assigned to maturity levels 2 and 3.

At maturity level 4 Subprocesses are selected that significantly contribute to overall process performance. These selected subprocesses are controlled using statistical and other quantitative techniques.

Quantitative objectives for quality and process performance are established and used as criteria in managing processes. Quantitative objectives are based on the needs of the customer, end users, organization, and process implementers. Quality and process performance are understood in statistical terms and are managed throughout the life of the processes.

For these processes, detailed measures of process performance are collected and statistically analyzed. Special causes of process variation are identified and, where appropriate, the sources of special causes are corrected to prevent future occurrences.

Quality and process performance measures are incorporated into the organization.s measurement repository to support fact-based decision making in the future.

A critical distinction between maturity level 3 and maturity level 4 is the predictability of process performance. At maturity level 4, the performance of processes is controlled using statistical and other quantitative techniques, and is quantitatively predictable. At maturity level 3, processes are only qualitatively predictable.

CMMI Maturity Level 5 – Optimizing

- CAR – Causal Analysis and Resolution
- OID – Organizational Innovation and Deployment

At maturity level 5, an organization has achieved all the **specific goals** of the process areas assigned to maturity levels 2, 3, 4, and 5 and the **generic goals** assigned to maturity levels 2 and 3.

Processes are continually improved based on a quantitative understanding of the common causes of variation inherent in processes.

Maturity level 5 focuses on continually improving process performance through both incremental and innovative technological improvements.

Quantitative process-improvement objectives for the organization are established, continually revised to reflect changing business objectives, and used as criteria in managing process improvement.

The effects of deployed process improvements are measured and evaluated against the quantitative process-improvement objectives. Both the defined processes and the organization's set of standard processes are targets of measurable improvement activities.

Optimizing processes that are agile and innovative depends on the participation of an empowered workforce aligned with the business values and objectives of the organization. The organization's ability to rapidly respond to changes and opportunities is enhanced by finding ways to accelerate and share learning. Improvement of the processes is inherently part of everybody's role, resulting in a cycle of continual improvement.

A critical distinction between maturity level 4 and maturity level 5 is the type of process variation addressed. At maturity level 4, processes are concerned with addressing special causes of process variation and providing statistical predictability of the results. Though processes may produce predictable results, the results may be insufficient to achieve the established objectives. At maturity level 5, processes are concerned with addressing common causes of process variation and changing the process (that is, shifting the mean of the process performance) to

improve process performance (while maintaining statistical predictability) to achieve the established quantitative process-improvement objectives.

Maturity Levels should not be skipped

Each maturity level provides a necessary foundation for effective implementation of processes at the next level.

- Higher level processes have less chance of success without the discipline provided by lower levels.
- The effect of innovation can be obscured in a noisy process.

Higher maturity level processes may be performed by organizations at lower maturity levels, with the risk of not being consistently applied in a crisis.

Process Patterns

As the software team moves through the software process they encounter problems. It would be very useful if solutions to these problems were readily available so that problems can be resolved quickly. Process-related problems which are encountered during software engineering work, it identifies the encountered problem and in which environment it is found, then it will suggest proven solutions to problem, they all are described by process pattern. By solving problems a software team can construct a process that best meets needs of a project.

Uses of the process pattern:-

At any level of abstraction, patterns can be defined. They can be used to describe a problem and solution associated with framework activity in some situations. While in other situations patterns can be used to describe a problem and solution associated with a complete process model.

Template: -

Pattern Name –
Meaningful name must be given to a pattern within context of software process (e.g. Technical Reviews).
Forces –
The issues that make problem visible and may affect its solution also environment in which pattern is encountered.

Type: -

It is of three types: -

1.Stage pattern –

Problems associated with a framework activity for process are described by stage pattern. Establishing Communication might be an example of a staged pattern. This pattern would incorporate task pattern Requirements Gathering and others.

2.Task-pattern –

Problems associated with a software engineering action or work task and relevant to successful software engineering practice (e.g., Requirements Gathering is a task pattern) are defined by task-pattern.

3.Phase pattern –

Even when the overall flow of activities is iterative in nature, it defines sequence of framework activities that occurs within process. Spiral Model or Prototyping might be an example of a phase pattern.

Initial Context:-

Conditions under which the pattern applies are described by initial context. Prior to the initiation of the pattern :

1. What organizational or term-related activities have already occurred?
2. Entry state for the process?
3. Software engineering information or project information already exists?

For example, the Planning pattern requires that

1. Collaborative communication has been established between customers and software engineers.
2. Successful completion of a number of task patterns for the communication pattern has occurred.
3. The project constraints, basic requirements, and the project scope are known.

Problem: -

Any specific problem is to be solved by pattern.

Solution –

Describes how to implement pattern successfully. This section describes how initial state of process is modified as a consequence of initiation of pattern.

Resulting Context: -

Once the pattern has been successfully implemented, it describes conditions. Upon completion of pattern:

1. Organizational or term-related activities must have occurred?
2. What should be the exit state for the process?
3. What software engineering information has been developed?

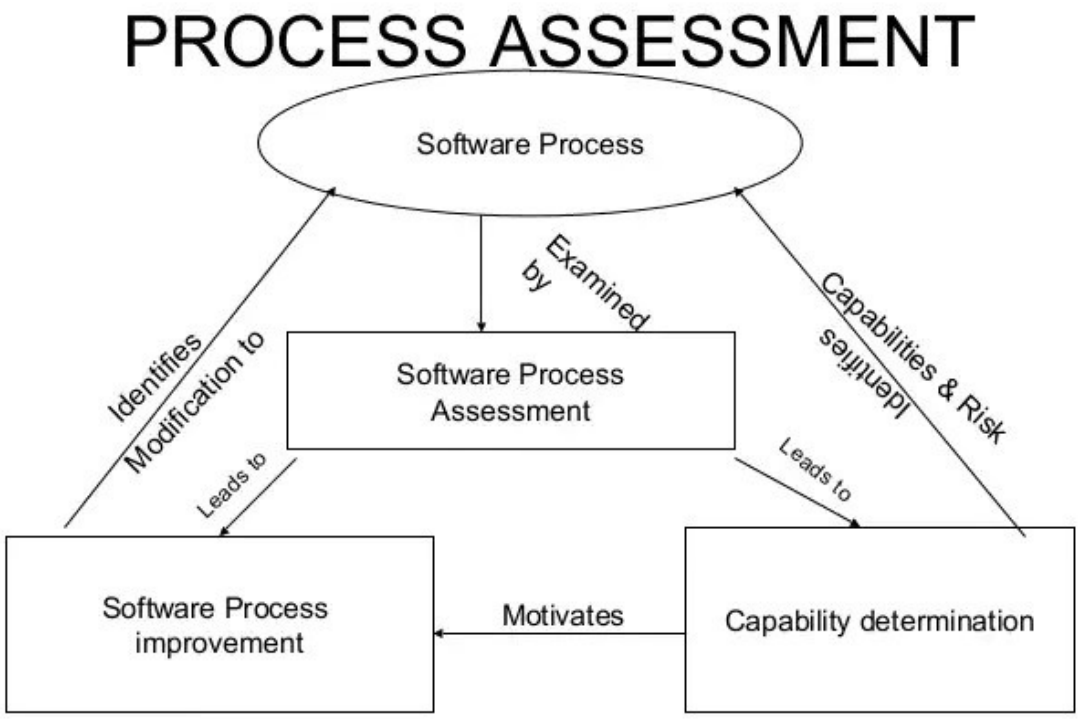
Related pattern: -
Provide a list of all process patterns that are directly related to this one. It can be represented n a hierarchy or in some other diagrammatic form.

Known uses and Examples –
In which the pattern is applicable, it indicates the specific instances. For example, communication is mandatory at the beginning of every software project, is recommended throughout the software project, and is mandatory once the deployment activity is underway.

Process Assessment

- Does not specify the quality of the software or whether the software will be delivered on time or will it stand up to the user requirements.

- It attempts to keep a check on the current state of the software process with the intention of improving it.



Approaches To Software Assessment: -

- Standard CMMI assessment (SCAMPI)
- CMM based appraisal for internal process improvement
- SPICE (ISO/IEC 15504)
- ISO 9001:2000 for software

Personal and Team Software Process: -

Personal software process

- PLANNING
- HIGH LEVEL DESIGN
- HIGH LEVEL DESIGN REVIEW
- DEVELOPMENT
- POSTMORTEM

Team software process
Goal of TSP

- Build self-directed teams
- Motivate the teams
- Acceptance of CMM level 5 behavior as normal to accelerate software process improvement
- Provide improvement guidance to high maturity organization

PROCESS MODELS

Prescriptive Process Models

The following framework activities are carried out irrespective of the process model chosen by the organization.

1. Communication
2. Planning
3. Modeling
4. Construction
5. Deployment

The name 'prescriptive' is given because the model prescribes a set of activities, actions, tasks, quality assurance and change the mechanism for every project.

There are three types of prescriptive process models. They are:

1. The Waterfall Model
2. Incremental Process model
3. RAD model

The waterfall model:-

- The waterfall model is also called as **'Linear sequential model'** or **'Classic life cycle model'.**
- In this model, each phase is fully completed before the beginning of the next phase.
- This model is used for the small projects.
- In this model, feedback is taken after each phase to ensure that the project is on the right path.
- Testing part starts only after the development is complete.

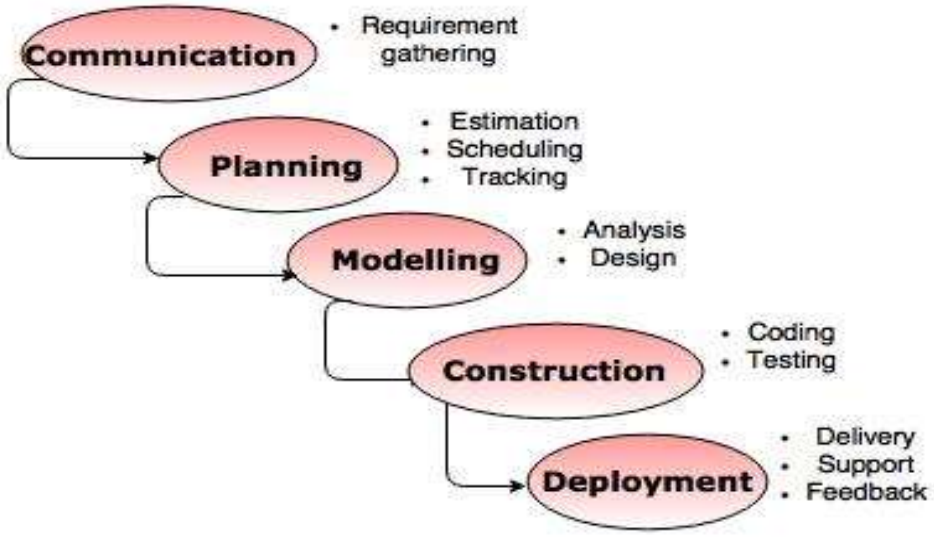


Fig. - The Waterfall model

NOTE: The description of the phases of the waterfall model is same as that of the process model.

An alternative design for 'linear sequential model' is as follows:

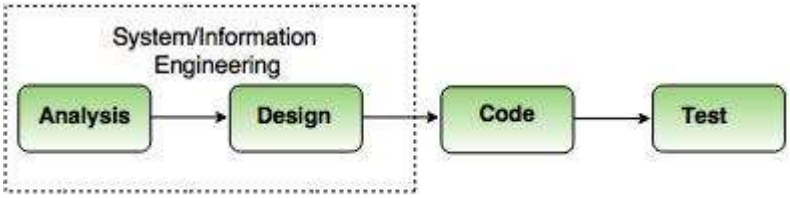


Fig. - The linear sequential model

Advantages of waterfall model:-

- The waterfall model is simple and easy to understand, implement, and use.
- All the requirements are known at the beginning of the project, hence it is easy to manage.
- It avoids overlapping of phases because each phase is completed at once.
- This model works for small projects because the requirements are understood very well.
- This model is preferred for those projects where the quality is more important as compared to the cost of the project.

Disadvantages of the waterfall model:-

- This model is not good for complex and object oriented projects.
- It is a poor model for long projects.
- The problems with this model are uncovered, until the software testing.
- The amount of risk is high.

1. Incremental Process Model:-

- The incremental model combines the elements of waterfall model and they are applied in an iterative fashion.
- The first increment in this model is generally a core product.
- Each increment builds the product and submits it to the customer for any suggested modifications.
- The next increment implements on the customer's suggestions and add additional requirements in the previous increment.
- This process is repeated until the product is finished.

  For example, the word-processing software is developed using the incremental model.

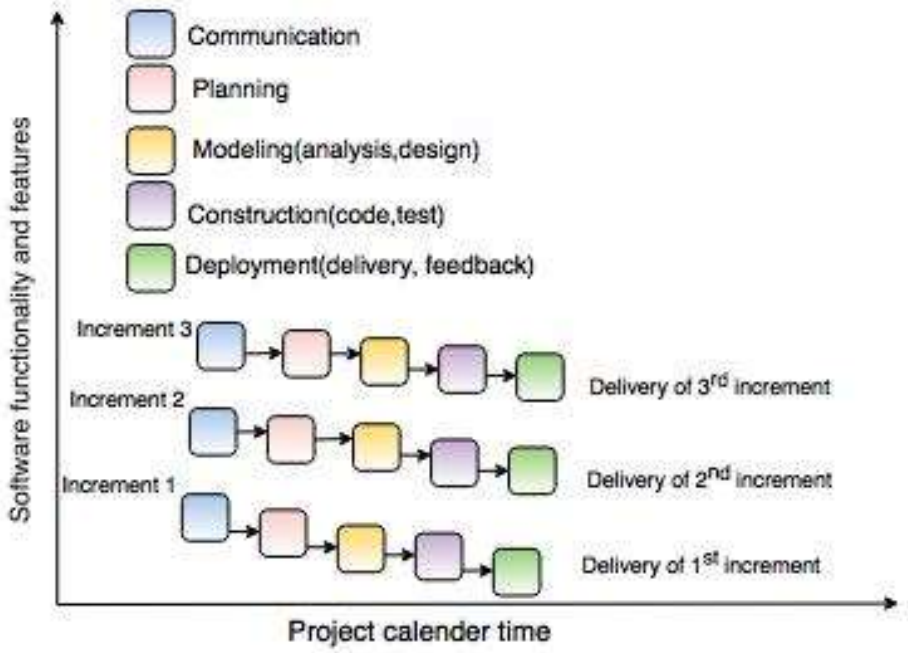


Fig. - Incremental Process Model

Advantages of incremental model:-

- This model is flexible because the cost of development is low and initial product delivery is faster.
- It is easier to test and debug during the smaller iteration.
- The working software generates quickly and early during the software life cycle.
- The customers can respond to its functionalities after every increment.

Disadvantages of the incremental model:-

- The cost of the final product may cross the cost estimated initially.
- This model requires a very clear and complete planning.
- The planning of design is required before the whole system is broken into small increments.
- The demands of customer for the additional functionalities after every increment causes problem during the system architecture.

3. RAD model:-

- RAD is a Rapid Application Development model.
- Using the RAD model, software product is developed in a short period of time.
- The initial activity starts with the communication between customer and developer.
- Planning depends upon the initial requirements and then the requirements are divided into groups.
- Planning is more important to work together on different modules.

The RAD model consist of following phases:-

1. Business Modeling

- Business modeling consists of the flow of information between various functions in the project.
- For example what type of information is produced by every function and which are the functions to handle that information.
- A complete business analysis should be performed to get the essential business information.

2. Data modeling

- The information in the business modeling phase is refined into the set of objects and it is essential for the business.
- The attributes of each object are identified and define the relationship between objects.

3. Process modeling

- The data objects defined in the data modeling phase are changed to fulfil the information flow to implement the business model.
- The process description is created for adding, modifying, deleting or retrieving a data object.

4. Application generation

- In the application generation phase, the actual system is built.

- To construct the software the automated tools are used.

5. Testing and turnover

- The prototypes are independently tested after each iteration so that the overall testing time is reduced.
- The data flow and the interfaces between all the components are are fully tested. Hence, most of the programming components are already tested.

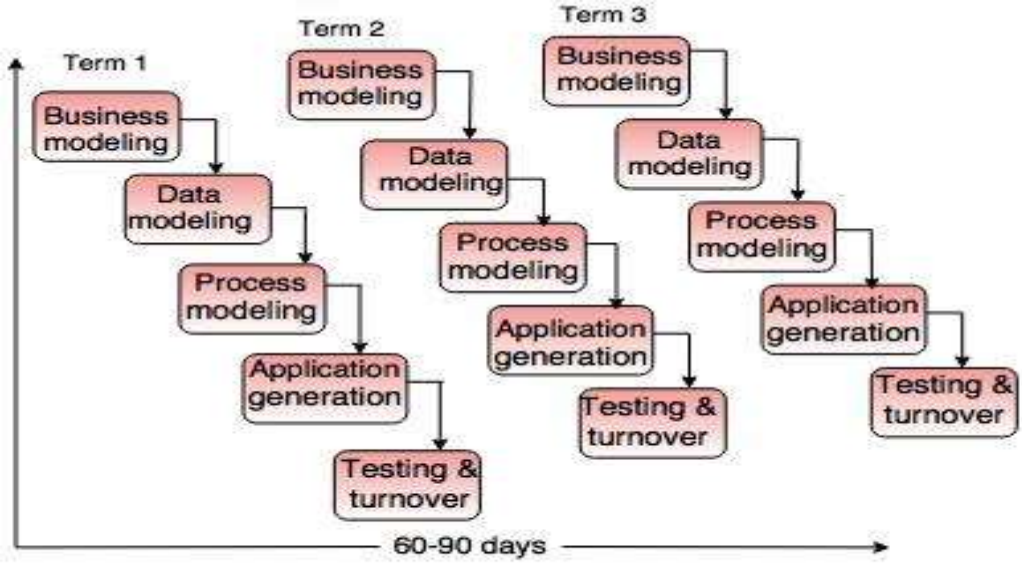


Fig. - RAD Model

Evolutionary Process Models

- Evolutionary models are iterative type models.
- They allow to develop more complete versions of the software.

Following are the evolutionary process models.

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

1. The Prototyping model:-

- Prototype is defined as first or preliminary form using which other forms are copied or derived.
- Prototype model is a set of general objectives for software.
- It does not identify the requirements like detailed input, output.
- It is software working model of limited functionality.
- In this model, working programs are quickly produced.

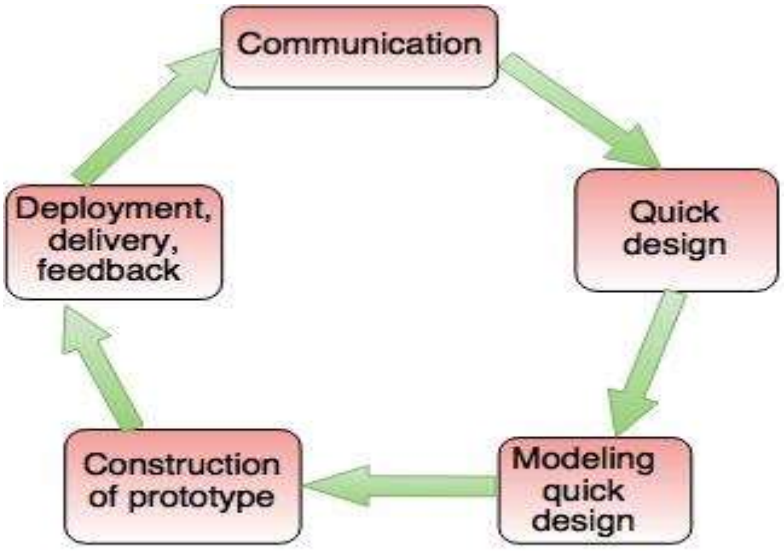


Fig. - The Prototyping Model

The different phases of Prototyping model are:

1. Communication

In this phase, developer and customer meet and discuss the overall objectives of the software.

2. Quick design

- Quick design is implemented when requirements are known.
- It includes only the important aspects like input and output format of the software.
- It focuses on those aspects which are visible to the user rather than the detailed plan.
- It helps to construct a prototype.

3. Modeling quick design

- This phase gives the clear idea about the development of software because the software is now built.
- It allows the developer to better understand the exact requirements.

4. Construction of prototype

The prototype is evaluated by the customer itself.

5. Deployment, delivery, feedback

- If the user is not satisfied with current prototype then it refines according to the requirements of the user.
- The process of refining the prototype is repeated until all the requirements of users are met.
- When the users are satisfied with the developed prototype then the system is developed on the basis of final prototype.

Advantages of Prototyping Model:-

- Prototype model need not know the detailed input, output, processes, adaptability of operating system and full machine interaction.
- In the development process of this model users are actively involved.
- The development process is the best platform to understand the system by the user.
- Errors are detected much earlier.
- Gives quick user feedback for better solutions.
- It identifies the missing functionality easily. It also identifies the confusing or difficult functions.

Disadvantages of Prototyping Model:-

- The client involvement is more and it is not always considered by the developer.
- It is a slow process because it takes more time for development.
- Many changes can disturb the rhythm of the development team.
- It is a thrown away prototype when the users are confused with it.

2. The Spiral model

- Spiral model is a risk driven process model.
- It is used for generating the software projects.
- In spiral model, an alternate solution is provided if the risk is found in the risk analysis, then alternate solutions are suggested and implemented.
- It is a combination of prototype and sequential model or waterfall model.
- In one iteration all activities are done, for large projects the output is small.

The framework activities of the spiral model are as shown in the following figure.

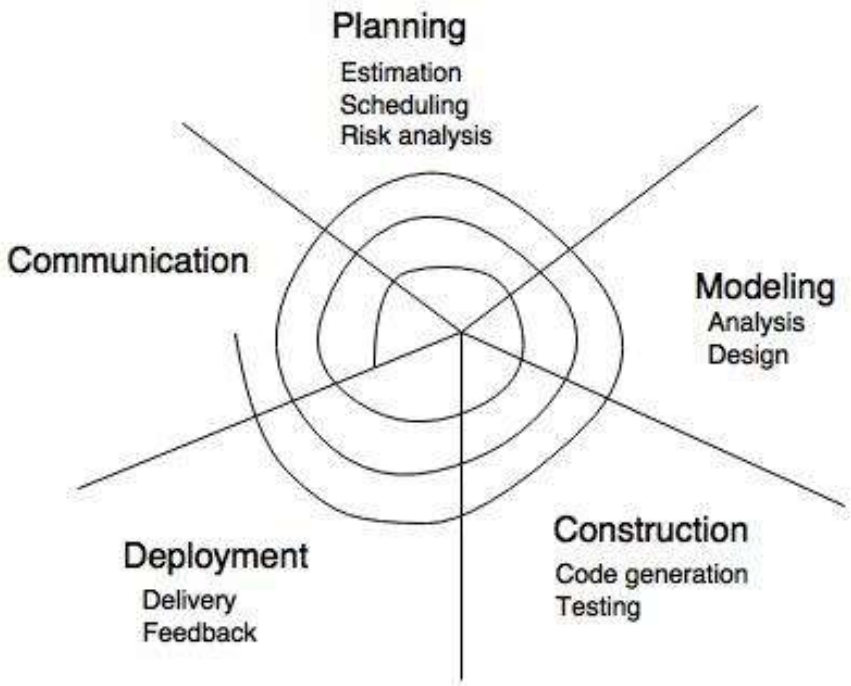


Fig. - The Spiral Model

NOTE: The description of the phases of the spiral model is same as that of the process model.

Advantages of Spiral Model:-

- It reduces high amount of risk.
- It is good for large and critical projects.
- It gives strong approval and documentation control.
- In spiral model, the software is produced early in the life cycle process.

Disadvantages of Spiral Model:-

- It can be costly to develop a software model.
- It is not used for small projects.

3. The concurrent development model

- The concurrent development model is called as concurrent model.
- The communication activity has completed in the first iteration and exits in the awaiting changes state.
- The modeling activity completed its initial communication and then go to the underdevelopment state.
- If the customer specifies the change in the requirement, then the modeling activity moves from the under development state into the awaiting change state.
- The concurrent process model activities moving from one state to another state.

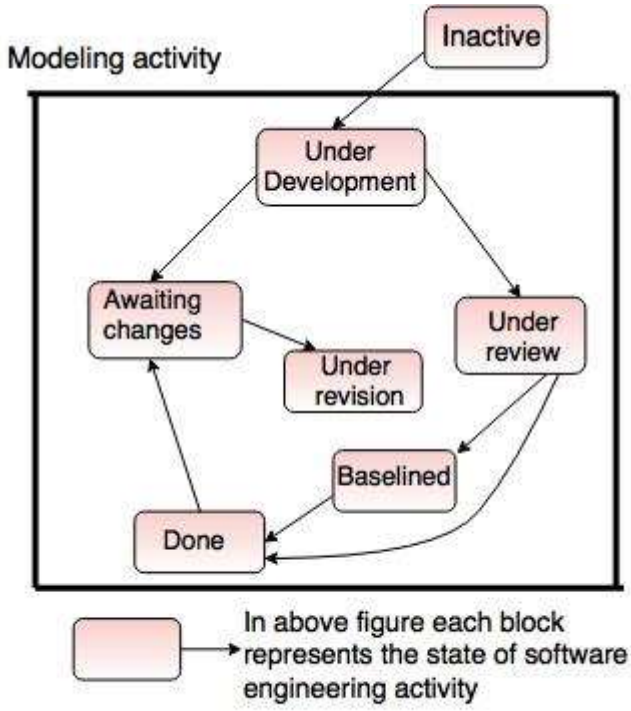


Fig. - One element of the concurrent process model

Advantages of the concurrent development model:-

- This model is applicable to all types of software development processes.
- It is easy for understanding and use.
- It gives immediate feedback from testing.
- It provides an accurate picture of the current state of a project.

Disadvantages of the concurrent development model:-

- It needs better communication between the team members. This may not be achieved all the time.
- It requires to remember the status of the different activities.

THE UNIFIED PROCESS:

The unified process (UP) is an attempt to draw on the best features and characteristics of conventional software process models, but characterize them in a way that implements many of the best principles of agile software development.

The Unified process recognizes the importance of customer communication and streamlined methods for describing the customer‘s view of a system. It emphasizes the important role of software architecture and

—helps the architect focus on the right goals, such as understandability, reliance to future changes, and reuse—. If suggests a process flow that is iterative and incremental, providing the evolutionary feel that is essential in modern software development.

A BRIEF HISTORY:

During the 1980s and into early 1990s, object-oriented (OO) methods and programming languages gained a widespread audience throughout the software engineering community. A wide variety of object- oriented analysis (OOA) and design (OOD) methods were proposed during the same time period.

During the early 1990s James Rumbaugh, Grady Booch, and IvalJacobsom began working on a —Unified method‖ that would combine the best features of each of OOD & OOA. The result was UML- a unified modeling language that contains a robust notation fot the modeling and development of OO systems.

By 1997, UML became an industry standard for object-oriented software development. At the same time, the Rational Corporation and other vendors developed automated tools to support UML methods.

Over the next few years, Jacobson, Rumbugh, and Booch developed the Unified process, a framework for object-oriented software engineering using UML. Today, the Unified process and UML are widely used on OO projects of all kinds. The iterative, incremental model proposed by the UP can and should be adapted to meet specific project needs.

Phases Of The Unified Process:

The **inception** phase of the UP encompasses both customer communication and planning activities. By collaborating with the customer and end-users, business requirements for the software are identified, a rough architecture for the system is proposed and a plan for the iterative, incremental nature of the ensuing project is developed.

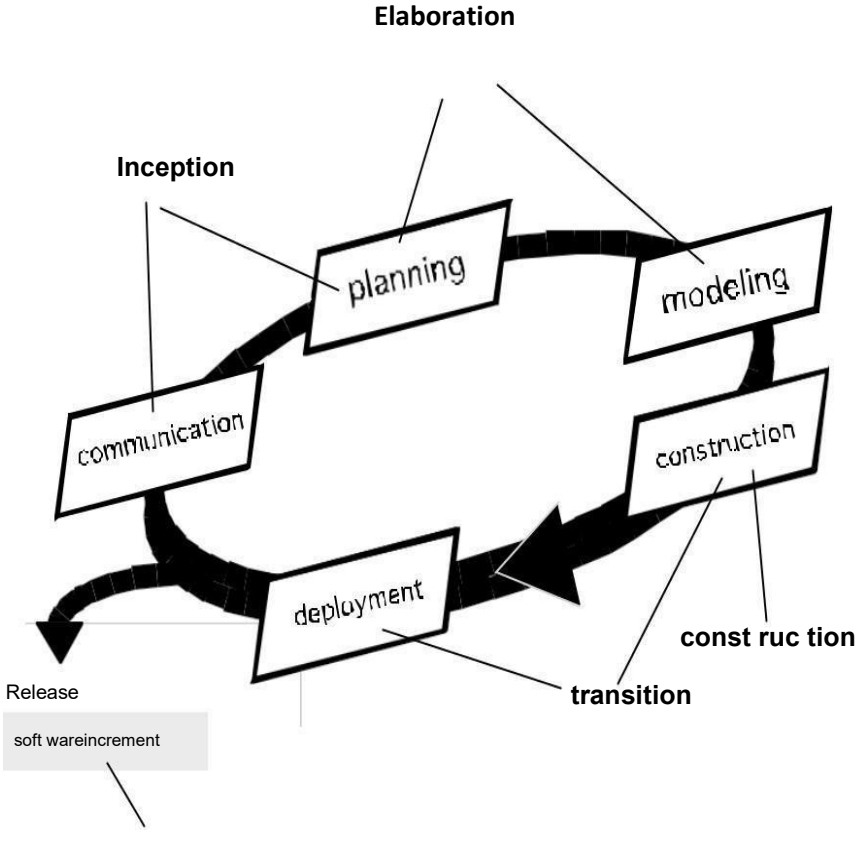
The **elaboration** phase encompasses the customer communication and modeling activities of the generic process model. Elaboration refines and expands the preliminary use-cases that were developed as part of the inception phase and expands the architectural

representation to include five different views of the software- the use-case model, the analysis model, the design model, the implementation model, and the deployment model.

The **construction** phase of the UP is identical to the construction activity defined for the generic software process. Using the architectural model as input, the construction phase develops or acquires the software components that will make each use-case operational for end-users. To accomplish this, analysis and design models that were started during the elaboration phase are completed to reflect the final version of the software increment.

The **transition** phase of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment activity. Software given to end-users for beta testing, and user feedback reports both defects and necessary changes.

The **production** phase of the UP coincides with the deployment activity of the generic process. During this phase, the on-going use of the software is monitored, support for the operating environment is provided, and defect reports and requests for changes are submitted and evaluated.



production

A software engineering workflow is distributed across all UP phases. In the context of UP, a *workflow* isanalogousto atask set. That is, aworkflow identifiesthe tasks required to accomplishanimportantsoftware engineering action and the work products that are produced as a consequence of successfullycompletingthetasks.

Prepared by
A.Sai Kumar
Assistant Professor
CSE Dept

Unit-2

Software Requirements

We should try to understand what sort of requirements may arise in the requirement elicitation phase and what kinds of requirements are expected from the software system.

Broadly software requirements should be categorized in two categories:

Functional Requirements:-

Requirements, which are related to functional aspect of software fall into this category.

They define functions and functionality within and from the software system.

Examples -

- ✓ Search option given to user to search from various invoices.
- ✓ User should be able to mail any report to management.
- ✓ Users can be divided into groups and groups can be given separate rights.
- ✓ Should comply business rules and administrative functions.
- ✓ Software is developed keeping downward compatibility intact.

Non-Functional Requirements:-

Requirements, which are not related to functional aspect of software, fall into this category. They are implicit or expected characteristics of software, which users make assumption of.

Non-functional requirements include -

- Security
- Logging
- Storage
- Configuration
- Performance
- Cost
- Interoperability
- Flexibility
- Disaster recovery
- Accessibility

Requirements are categorized logically as

- **Must Have** : Software cannot be said operational without them.
- **Should have** : Enhancing the functionality of software.
- **Could have** : Software can still properly function with these requirements.
- **Wish list** : These requirements do not map to any objectives of software.

While developing software, 'Must have' must be implemented, 'Should have' is a matter of debate with stakeholders and negation, whereas 'could have' and 'wish list' can be kept for software updates.

User Interface requirements

UI is an important part of any software or hardware or hybrid system. A software is widely accepted if it is -

- easy to operate
- quick in response
- effectively handling operational errors
- providing simple yet consistent user interface

User acceptance majorly depends upon how user can use the software. UI is the only way for users to perceive the system. A well performing software system must also be equipped with attractive, clear, consistent and responsive user interface. Otherwise the functionalities of software system can not be used in convenient way. A system is said be good if it provides means to use it efficiently. User interface requirements are briefly mentioned below -

- Content presentation
- Easy Navigation
- Simple interface
- Responsive
- Consistent UI elements
- Feedback mechanism
- Default settings
- Purposeful layout
- Strategical use of color and texture.
- Provide help information
- User centric approach
- Group based view settings.

Software System Analyst:-

System analyst in an IT organization is a person, who analyzes the requirement of proposed system and ensures that requirements are conceived and documented properly & correctly. Role of an analyst starts during Software Analysis Phase of SDLC. It is the responsibility of analyst to make sure that the developed software meets the requirements of the client.

System Analysts have the following responsibilities:

- ✓ Analyzing and understanding requirements of intended software
- ✓ Understanding how the project will contribute in the organization objectives
- ✓ Identify sources of requirement
- ✓ Validation of requirement
- ✓ Develop and implement requirement management plan

- ✓ Documentation of business, technical, process and product requirements
- ✓ Coordination with clients to prioritize requirements and remove and ambiguity
- ✓ Finalizing acceptance criteria with client and other stakeholders

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.

According to Tom DeMarco, a (Software Engineer), "You cannot control what you cannot measure." By his saying, it is very clear how important software measures are.

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.

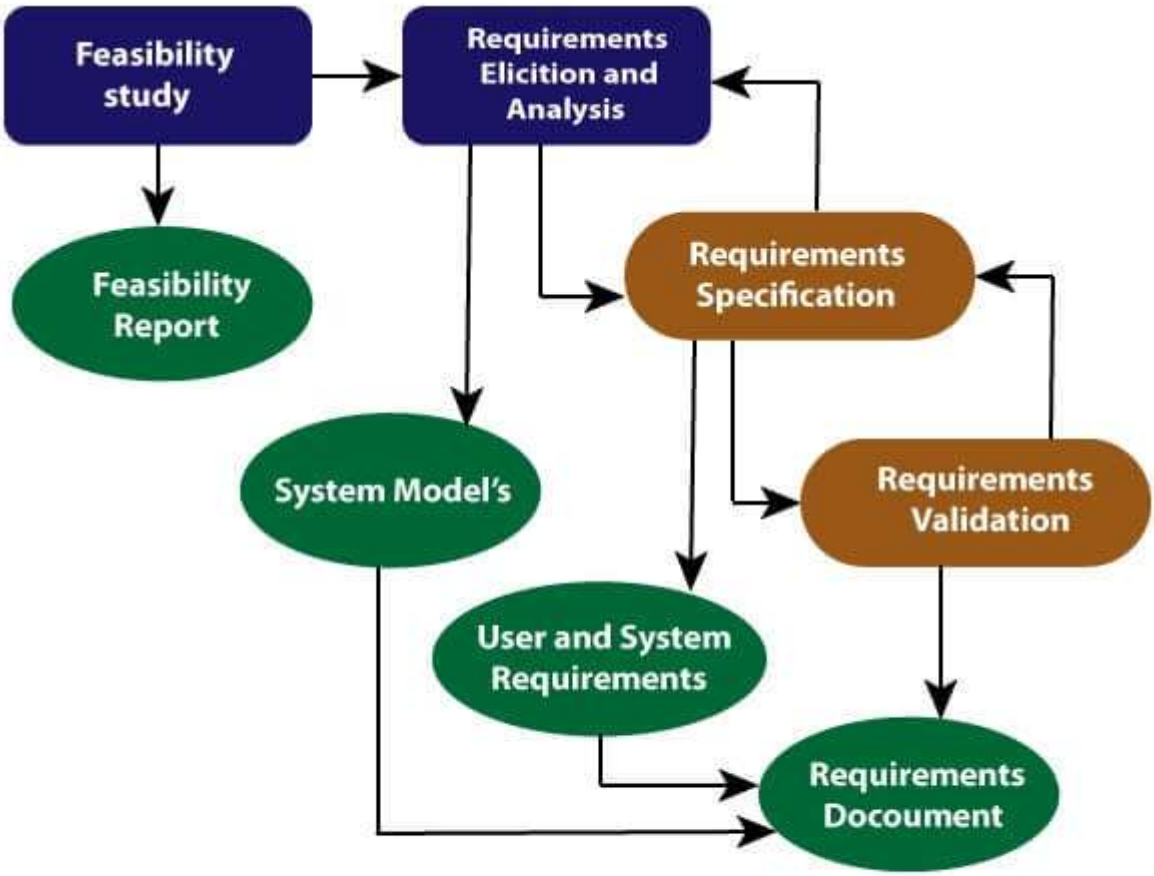
Requirement Engineering

Requirements engineering (RE) refers to the process of defining, documenting, and maintaining requirements in the engineering design process. Requirement engineering provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system. Thus, requirement engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints.

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.

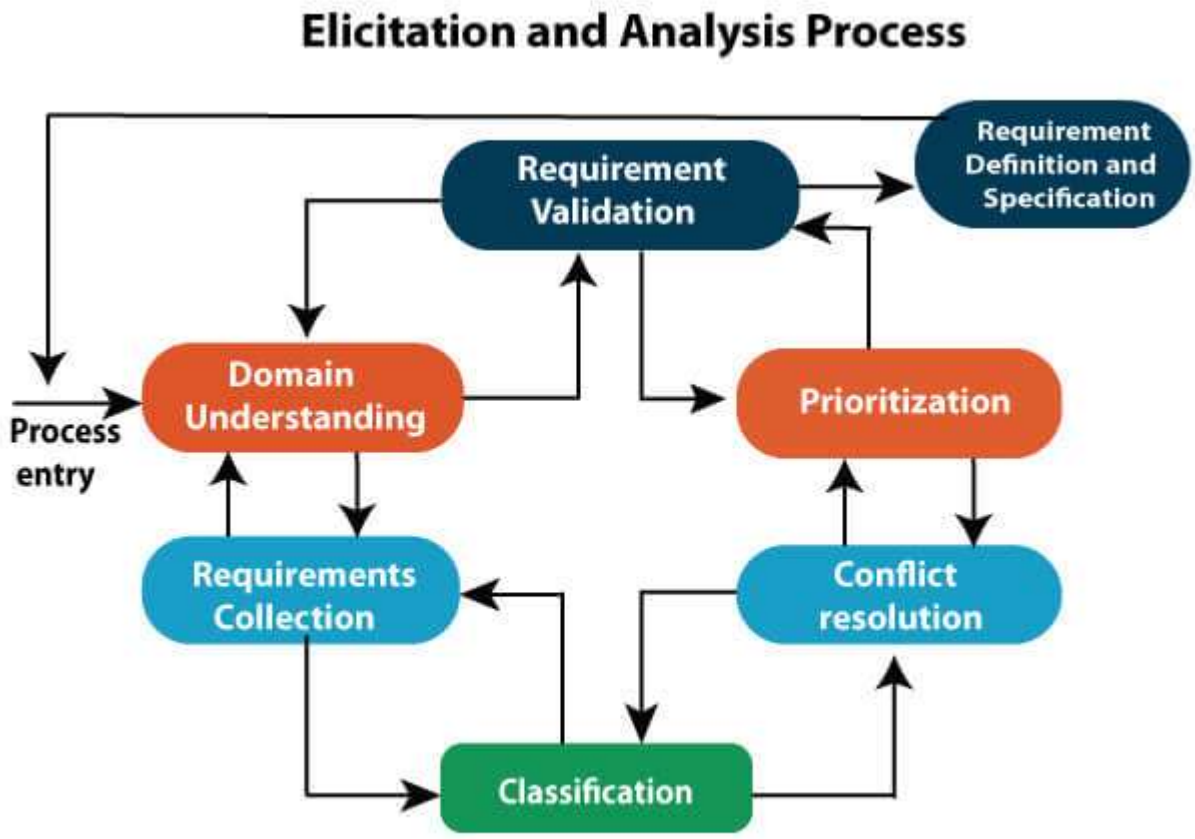
2. Requirement Elicitation and Analysis: -

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.

5. Software Requirement Management

- ❖ Requirement management is the process of managing changing requirements during the requirements engineering process and system development.
- ❖ New requirements emerge during the process as business needs a change, and a better understanding of the system is developed.
- ❖ The priority of requirements from different viewpoints changes during development process.
- ❖ The business and technical environment of the system changes during the development.

Prerequisite of Software requirements: -

Collection of software requirements is the basis of the entire software development project. Hence, they should be clear, correct, and well-defined.

A complete Software Requirement Specifications should be:

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

System Models

System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.

Different models present the system from different perspectives

- Behavioural perspective showing the behaviour of the system;
- Structural perspective showing the system or data architecture.

Model types

Data processing model showing how the data is processed at different stages. Composition model showing how entities are composed of other entities.

Architectural model showing principal sub-systems.

Classification model showing how entities have common characteristics. Stimulus/response model showing the system‘s reaction to events.

System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system. It is about representing a system using some kind of graphical notation, which is now almost always based on notations in the **Unified Modeling Language** (**UML**). Models help the analyst to understand the functionality of the system; they are used to communicate with customers.

Models can explain the system from **different perspectives**:

- An **external** perspective, where you model the context or environment of the system.
- An **interaction** perspective, where you model the interactions between a system and its environment, or between the components of a system.
- A **structural** perspective, where you model the organization of a system or the structure of the data that is processed by the system.
- A **behavioral** perspective, where you model the dynamic behavior of the system and how it responds to events.

Five types of UML diagrams that are the most useful for system modeling:

- **Activity** diagrams, which show the activities involved in a process or in data processing.
- **Use case** diagrams, which show the interactions between a system and its environment.
- **Sequence** diagrams, which show interactions between actors and the system and between system components.
- **Class** diagrams, which show the object classes in the system and the associations between these classes.
- **State** diagrams, which show how the system reacts to internal and external events.

Models of both new and existing system are used during **requirements engineering**. Models of the **existing systems** help clarify what the existing system does and can be used as a basis for discussing its strengths and weaknesses. These then lead to requirements for the new system. Models of the **new system** are used during requirements engineering to help explain the proposed requirements to other system stakeholders. Engineers use these models to discuss design proposals and to document the system for implementation.

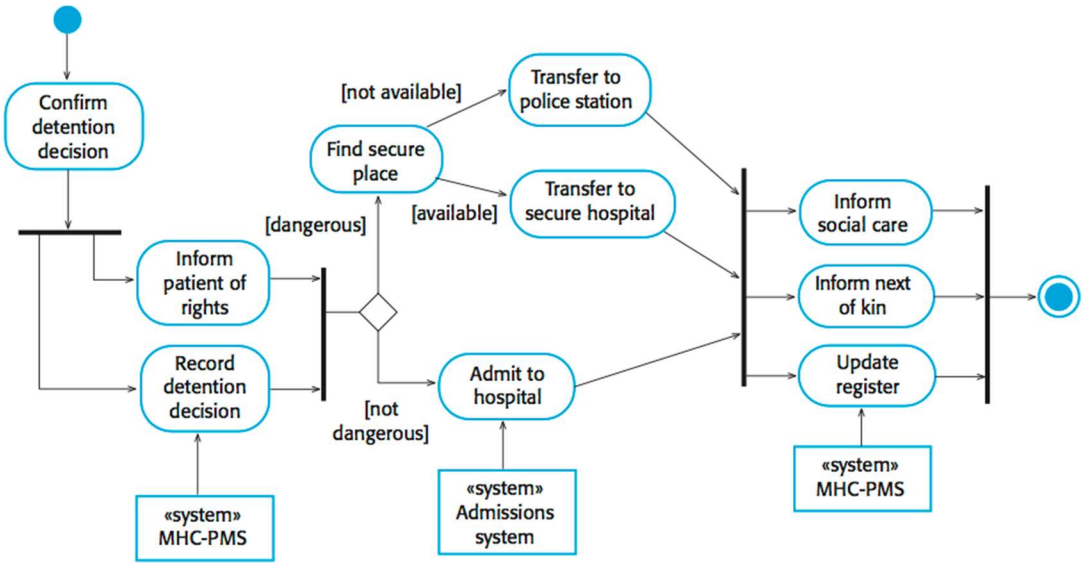
Context and process models

Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries. Social and organizational concerns may affect the decision on where to position system boundaries. Architectural models show the system and its relationship with other systems.

System boundaries are established to define what is inside and what is outside the system. They show other systems that are used or depend on the system being developed. The position of the system boundary has a profound effect on the system requirements. Defining a system boundary is a political judgment since there may be pressures to develop system boundaries that increase/decrease the influence or workload of different parts of an organization.

Context models simply show the other systems in the environment, not how the system being developed is used in that environment. **Process models** reveal how the system being developed is used in broader business processes. UML activity diagrams may be used to define business process models.

The example below shows a UML **activity diagram** describing the process of involuntary detention and the role of MHC-PMS (mental healthcare patient management system) in it.



Interaction models

Types of interactions that can be represented in a model:

- Modeling **user interaction** is important as it helps to identify user requirements.
- Modeling **system-to-system interaction** highlights the communication problems that may arise.
- Modeling **component interaction** helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.

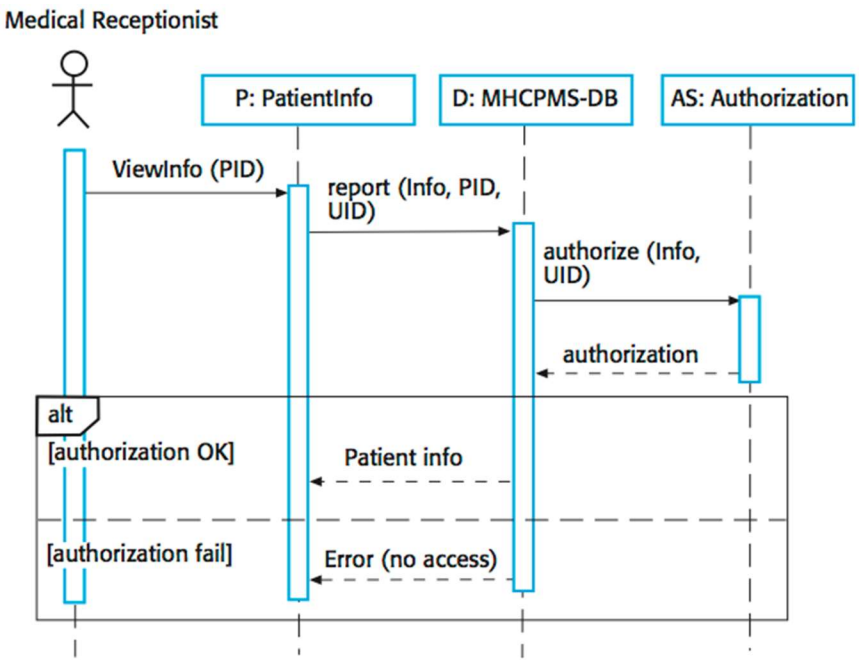
Use cases were developed originally to support requirements elicitation and now incorporated into the UML. Each use case represents a discrete task that involves external interaction with a system. Actors in a use case may be people or other systems. Use cases can be represented using a UML use case diagram and in a more detailed textual/tabular format.

Simple use case diagram:

Use case description in a tabular format:

Use case title	Transfer data
Description	A receptionist may transfer data from the MHC-PMS to a general patient record database that is maintained by a health authority. The information transferred may either be updated personal information (address, phone number, etc.) or a summary of the patient's diagnosis and treatment.
Actor(s)	Medical receptionist, patient records system (PRS)
Preconditions	Patient data has been collected (personal information, treatment summary); The receptionist must have appropriate security permissions to access the patient information and the PRS.
Postconditions	PRS has been updated
Main success scenario	1. Receptionist selects the "Transfer data" option from the menu. 2. PRS verifies the security credentials of the receptionist. 3. Data is transferred. 4. PRS has been updated.
Extensions	2a. The receptionist does not have the necessary security credentials. 2a.1. An error message is displayed. 2a.2. The receptionist backs out of the use case.

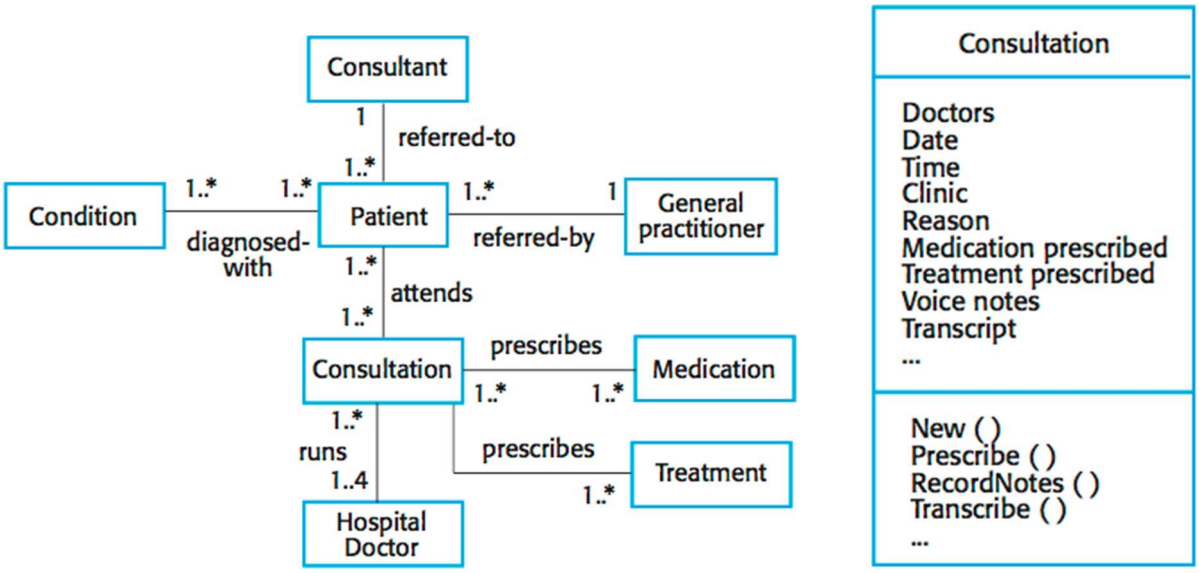
UML **sequence diagrams** are used to model the interactions between the actors and the objects within a system. A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance. The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these. Interactions between objects are indicated by annotated arrows.



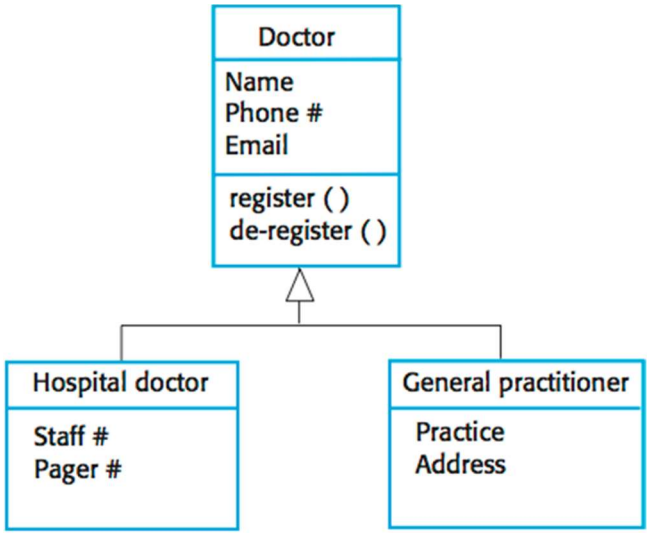
Structural models

Structural models of software display the organization of a system in terms of the components that make up that system and their relationships. Structural models may be **static** models, which show the structure of the system design, or **dynamic** models, which show the organization of the system when it is executing. You create structural models of a system when you are discussing and designing the system architecture.

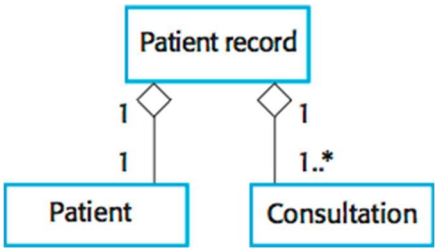
UML **class diagrams** are used when developing an object-oriented system model to show the classes in a system and the associations between these classes. An object class can be thought of as a general definition of one kind of system object. An association is a link between classes that indicates that there is some relationship between these classes. When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.



Generalization is an everyday technique that we use to manage complexity. In modeling systems, it is often useful to examine the classes in a system to see if there is scope for generalization. In object-oriented languages, such as Java, generalization is implemented using the class **inheritance** mechanisms built into the language. In a generalization, the attributes and operations associated with higher-level classes are also associated with the lower-level classes. The lower-level classes are subclasses inherit the attributes and operations from their superclasses. These lower-level classes then add more specific attributes and operations.



An **aggregation** model shows how classes that are collections are composed of other classes. Aggregation models are similar to the part-of relationship in semantic data models.



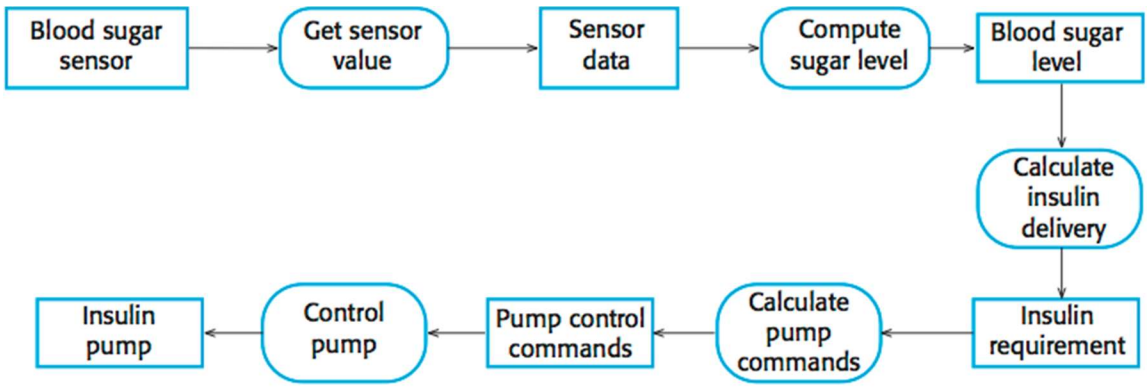
Behavioral models

Behavioral models are models of the dynamic behavior of a system as it is executing. They show what happens or what is supposed to happen when a system responds to a stimulus from its environment. Two types of stimuli:

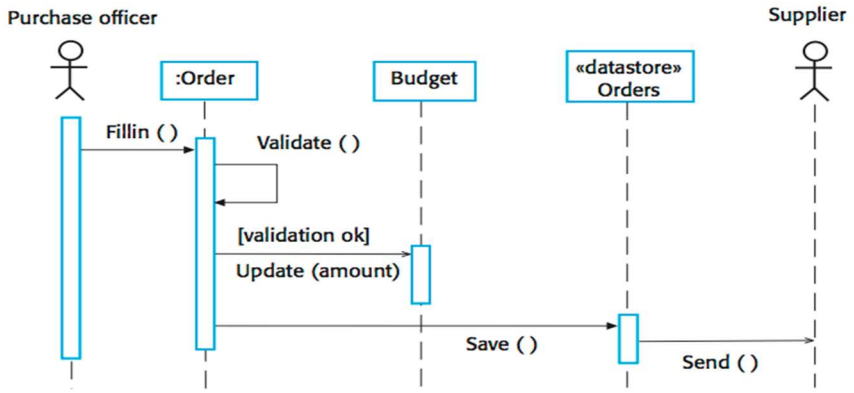
- Some **data** arrives that has to be processed by the system.
- Some **event** happens that triggers system processing. Events may have associated data, although this is not always the case.

Many business systems are data-processing systems that are primarily driven by data. They are controlled by the data input to the system, with relatively little external event processing. **Data-driven models** show the sequence of actions involved in processing input data and generating an associated output. They are particularly useful during the analysis of requirements as they can be used to show end-to-end processing in a system. Data-driven models can be created using

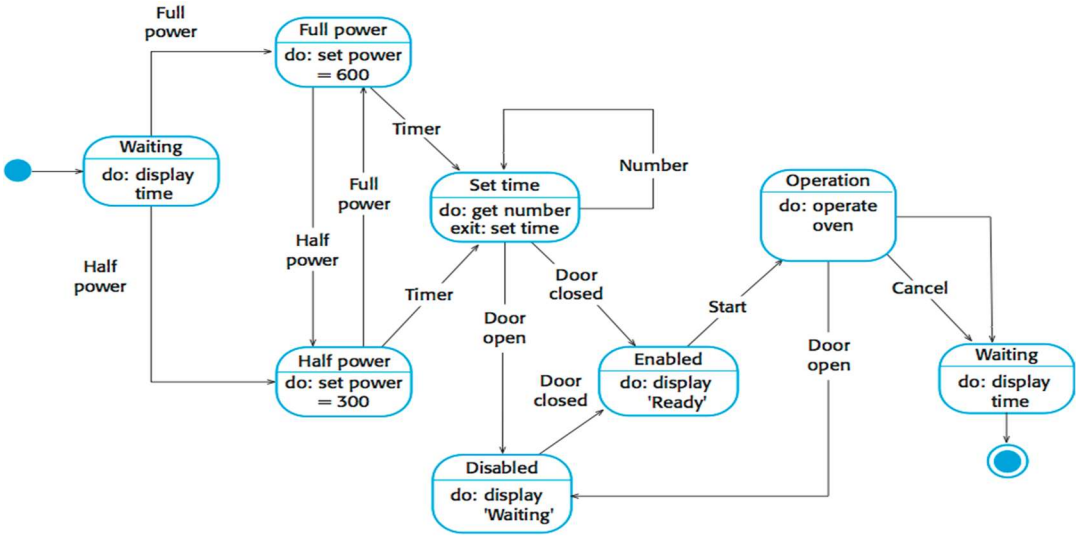
UML **activity diagrams**:



Data-driven models can also be created using UML **sequence diagrams**:



Real-time systems are often event-driven, with minimal data processing. For example, a landline phone switching system responds to events such as 'receiver off hook' by generating a dial tone. **Event-driven models** shows how a system responds to external and internal events. It is based on the assumption that a system has a finite number of states and that events (stimuli) may cause a transition from one state to another. Event-driven models can be created using UML **state diagrams**:



Unit-3

Software process designing concepts

Introduction to design process

- The main aim of design engineering is to generate a model which shows firmness, delight and commodity.
- Software design is an iterative process through which requirements are translated into the blueprint for building the software.

Software quality guidelines

- A design is generated using the recognizable architectural styles and compose a good design characteristic of components and it is implemented in evolutionary manner for testing.
- A design of the software must be modular i.e the software must be logically partitioned into elements.
- In design, the representation of data , architecture, interface and components should be distinct.
- A design must carry appropriate data structure and recognizable data patterns.
- Design components must show the independent functional characteristic.
- A design creates an interface that reduce the complexity of connections between the components.
- A design must be derived using the repeatable method.
- The notations should be use in design which can effectively communicates its meaning.

Quality attributes

The attributes of design name as 'FURPS' are as follows:

Functionality:

It evaluates the feature set and capabilities of the program.

Usability:

It is accessed by considering the factors such as human factor, overall aesthetics, consistency and documentation.

Reliability:

It is evaluated by measuring parameters like frequency and security of failure, output result accuracy, the mean-time-to-failure(MTTF), recovery from failure and the the program predictability.

Performance:

It is measured by considering processing speed, response time, resource consumption, throughput and efficiency.

Supportability:

- It combines the ability to extend the program, adaptability, serviceability. These three term defines the maintainability.
- Testability, compatibility and configurability are the terms using which a system can be easily installed and found the problem easily.
- Supportability also consists of more attributes such as compatibility, extensibility, fault tolerance, modularity, reusability, robustness, security, portability, scalability.

Design concepts

The set of fundamental software design concepts are as follows:

1. Abstraction

- A solution is stated in large terms using the language of the problem environment at the highest level abstraction.
- The lower level of abstraction provides a more detail description of the solution.
- A sequence of instruction that contain a specific and limited function refers in a procedural abstraction.

- A collection of data that describes a data object is a data abstraction.

2. Architecture

- The complete structure of the software is known as software architecture.
- Structure provides conceptual integrity for a system in a number of ways.
- The architecture is the structure of program modules where they interact with each other in a specialized way.
- The components use the structure of data.
- The aim of the software design is to obtain an architectural framework of a system.
- The more detailed design activities are conducted from the framework.

3. Patterns

A design pattern describes a design structure and that structure solves a particular design problem in a specified content.

4. Modularity

- A software is separately divided into name and addressable components. Sometime they are called as modules which integrate to satisfy the problem requirements.
- Modularity is the single attribute of a software that permits a program to be managed easily.

5. Information hiding

Modules must be specified and designed so that the information like algorithm and data presented in a module is not accessible for other modules not requiring that information.

6. Functional independence

- The functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.
- The functional independence is accessed using two criteria i.e Cohesion and coupling.

Cohesion

- Cohesion is an extension of the information hiding concept.
- A cohesive module performs a single task and it requires a small interaction with the other components in other parts of the program.

Coupling

Coupling is an indication of interconnection between modules in a structure of software.

7. Refinement

- Refinement is a top-down design approach.
- It is a process of elaboration.
- A program is established for refining levels of procedural details.
- A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statement are reached.

8. Refactoring

- It is a reorganization technique which simplifies the design of components without changing its function behaviour.
- Refactoring is the process of changing the software system in a way that it does not change the external behaviour of the code still improves its internal structure.

9. Design classes

- The model of software is defined as a set of design classes.
- Every class describes the elements of problem domain and that focus on features of the problem which are user visible.

Design concept in Software Engineering

- Object Oriented is a popular design approach for analyzing and designing an application.
- Most of the languages like C++, Java, .net are use object oriented design concept.
- Object-oriented concepts are used in the design methods such as classes, objects, polymorphism, encapsulation, inheritance, dynamic binding, information hiding, interface, constructor, destructor.
- The main advantage of object oriented design is that improving the software development and maintainability.
- Another advantage is that faster and low cost development, and creates a high quality software.
- The disadvantage of the object-oriented design is that larger program size and it is not suitable for all types of program.

Design classes

- A set of design classes refined the analysis class by providing design details.

There are five different types of design classes and each type represents the layer of the design architecture these are as follows:

1. User interface classes

- These classes are designed for Human Computer Interaction(HCI).
- These interface classes define all abstraction which is required for Human Computer Interaction(HCI).

2. Business domain classes

- These classes are commonly refinements of the analysis classes.
- These classes are recognized as attributes and methods which are required to implement the elements of the business domain.

3. Process classes

It implement the lower level business abstraction which is needed to completely manage the business domain class.

4. Persistence classes

It shows data stores that will persist behind the execution of the software.

5. System Classes

System classes implement software management and control functions that allow to operate and communicate in computing environment and outside world.

Design class characteristics

The characteristics of well formed designed class are as follows:

1. Complete and sufficient

A design class must be the total encapsulation of all attributes and methods which are required to exist for the class.

2. Primitiveness

- The method in the design class should fulfil one service for the class.
- If service is implemented with a method then the class should not provide another way to fulfil same thing.

3. High cohesion

- A cohesion design class has a small and focused set of responsibilities.
- For implementing the set of responsibilities the design classes are applied single-mindedly to the methods and attribute.

4. Low-coupling

- All the design classes should collaborate with each other in a design model.
- The minimum acceptable of collaboration must be kept in this model.
- If a design model is highly coupled then the system is difficult to implement, to test and to maintain over time.

Software design model elements

Following are the types of design elements:

1. Data design elements

- The data design element produced a model of data that represent a high level of abstraction.
- This model is then more refined into more implementation specific representation which is processed by the computer based system.
- The structure of data is the most important part of the software design.

2. Architectural design elements

- The architecture design elements provides us overall view of the system.
- The architectural design element is generally represented as a set of interconnected subsystem that are derived from analysis packages in the requirement model.

The architecture model is derived from following sources:

- The information about the application domain to built the software.

- Requirement model elements like data flow diagram or analysis classes, relationship and collaboration between them.
- The architectural style and pattern as per availability.

3. Interface design elements

- The interface design elements for software represents the information flow within it and out of the system.
- They communicate between the components defined as part of architecture.

Following are the important elements of the interface design:

1. The user interface
2. The external interface to the other systems, networks etc.
3. The internal interface between various components.

4. Component level diagram elements

- The component level design for software is similar to the set of detailed specification of each room in a house.
- The component level design for the software completely describes the internal details of the each software component.
- The processing of data structure occurs in a component and an interface which allows all the component operations.
- In a context of object-oriented software engineering, a component shown in a UML diagram.
- The UML diagram is used to represent the processing logic.

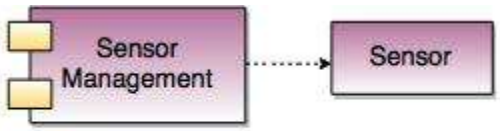


Fig. - UML component diagram for sensor managemnet

5. Deployment level design elements

- The deployment level design element shows the software functionality and subsystem that allocated in the physical computing environment which support the software.

- Following figure shows three computing environment as shown. These are the personal computer, the CPI server and the Control panel.

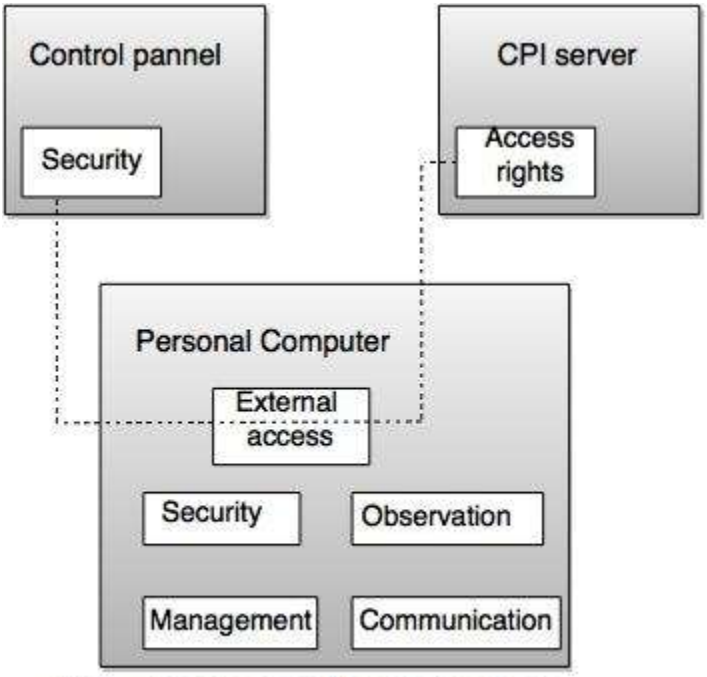


Fig. - Deployment level diagram

THE DESIGN MODEL:

The design model can be viewed into different dimensions.

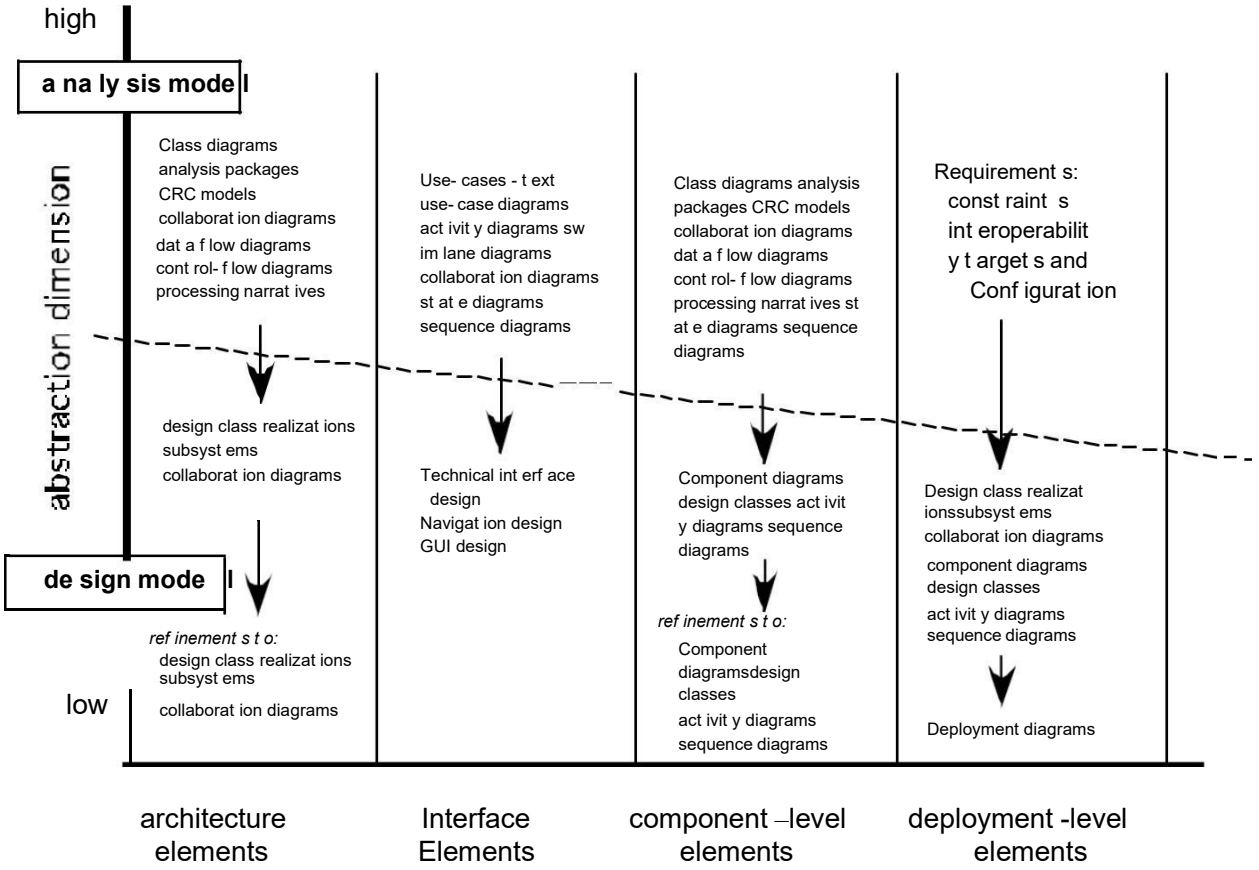
The process dimension indicates the evolution of the design model as design tasks are executed as a part of the software process.

The abstraction dimension represents the level of detail as each element of the analysis model is transformed into a design equivalent and then refined iteratively.

The elements of the design model use many of the same UML diagrams that were used in the analysis model. The difference is that these diagrams are refined and elaborated as a path of design; more implementation- specific detail is provided, and architectural structure and style,

components that reside within the architecture, and the interface between the components and with the outside world are all emphasized.

It is important to mention however, that model elements noted along the horizontal axis are not always developed in a sequential fashion. In most cases preliminary architectural design sets the stage and is followed by interface design and component-level design, which often occur in parallel. The deployment model us usually delayed until the design has been fully developed.


high
a na ly sis mode l
abstraction dimension
Class diagrams
analysis packages
CRC models
collaborat ion diagrams
dat a f low diagrams
cont rol- f low diagrams
processing narrat ives
design class realizat ions
subsyst ems
collaborat ion diagrams
de sign mode l
ref inement s t o:
design class realizat ions
subsyst ems
collaborat ion diagrams
low
Use- cases - t ext
use- case diagrams
act ivit y diagrams sw
im lane diagrams
collaborat ion diagrams
st at e diagrams
sequence diagrams
Technical int erf ace
design
Navigat ion design
GUI design
Class diagrams analysis
packages CRC models
collaborat ion diagrams
dat a f low diagrams
cont rol- f low diagrams
processing narrat ives st
at e diagrams sequence
diagrams
Component diagrams
design classes act ivit
y diagrams sequence
diagrams
ref inement s t o:
Component
diagramsdesign
classes
act ivit y diagrams
sequence diagrams
Requirement s:
const raint s
int eroperabilit
y t arget s and
Conf igurat ion
Design class realizat
ionssubsyst ems
collaborat ion diagrams
component diagrams
design classes
act ivit y diagrams
sequence diagrams
Deployment diagrams
architecture
elements
Interface
Elements
component –level
elements
deployment -level
elements

Process Dimension

Data design sometimes referred to as data architecting creates a model of data and/or information that is represented at a high level of abstraction. This data model is then refined into progressively more implementation-specific representations that can be processed by the computer-based system.

The structure of data has always been an important part of software design.

At the **program component level**, the design of data structures and the associated algorithms required to manipulate them is essential to the criterion of high-quality applications.

At **the application level**, the translation of a data model into a database is pivotal to achieving the business objectives of a system.

At the **business leve**l, the collection of information stored in disparate databases and reorganized into a

―data warehouse‖ enables data mining or knowledge discovery that can have an impact on the success of the business itself.

Architectural design elements:

The *architectural design* for software is the equivalent to the floor plan of a house. The architectural model is derived from three sources.

Information about the application domain for the software to be built.

Specific analysis model elements such as data flow diagrams or analysis classes, their relationships and collaborations for the problem at hand, and

The availability of architectural patterns

Interface design elements:

The *interface design* for software is the equivalent to a set of detailed drawings for the doors, windows, and external utilities of a house.

The interface design elements for software tell how information flows into and out of the system and how it is communicated among the components defined as part of the architecture. There are 3 important elements of interface design:

The user interface (UI);

External interfaces to other systems, devices, networks, or other produces or consumers of information; and

Internal interfaces between various design components.

These interface design elements allow the software to communicated externally and enable internal communication and collaboration among the components that populate the software architecture.

UI design is a major software engineering action.

The design of a UI incorporates aesthetic elements (e.g., layout, color, graphics, interaction mechanisms), ergonomic elements (e.g., information layout and placement, metaphors, UI navigation), and technical elements (e.g., UI patterns, reusable components). In general, the UI is a unique subsystem within the overall application architecture.

The design of external interfaces requires definitive information about the entity to which information is sent or received. The design of external interfaces should incorporate error checking and appropriated security features.

UML defines an *interface* in the following manner:‖an interface is a specifier for the externally-visible operations of a class, component, or other classifier without specification of internal structure.‖

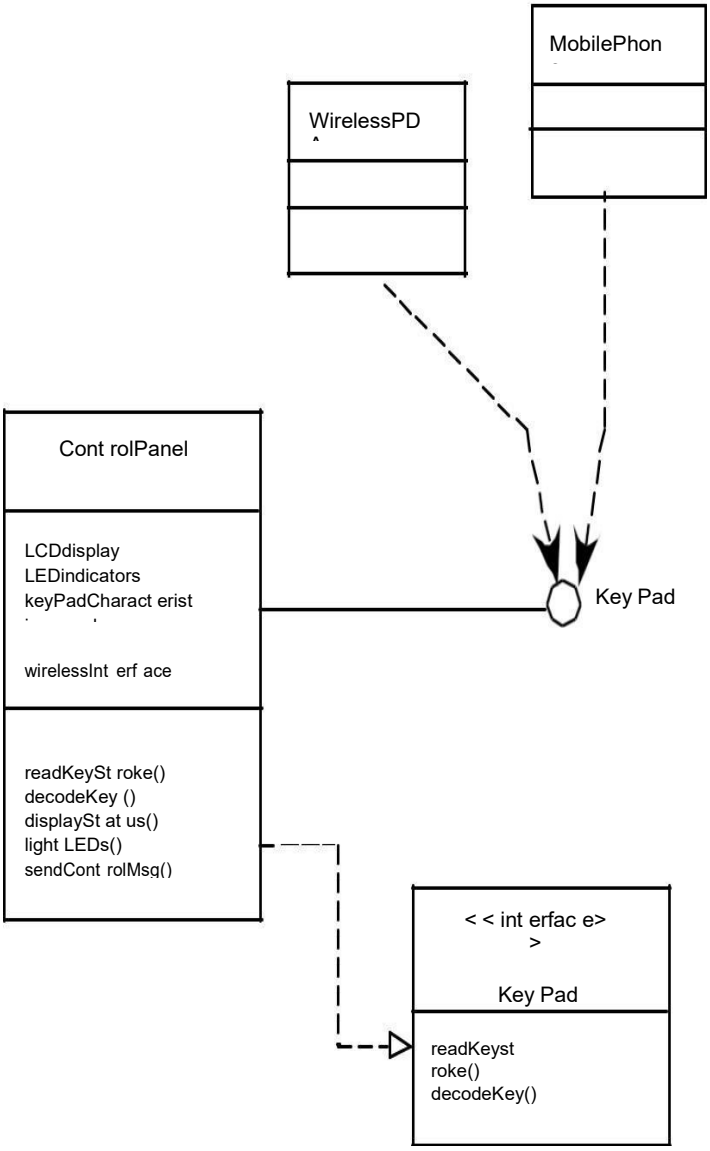
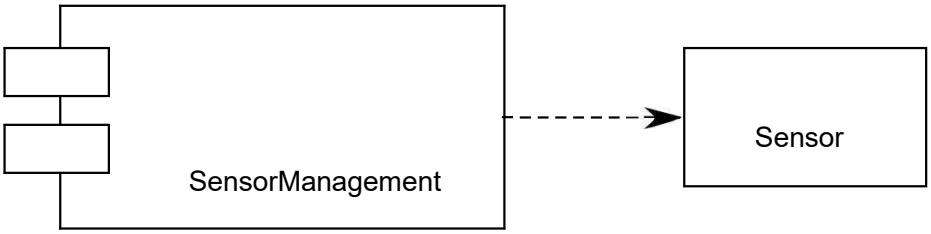


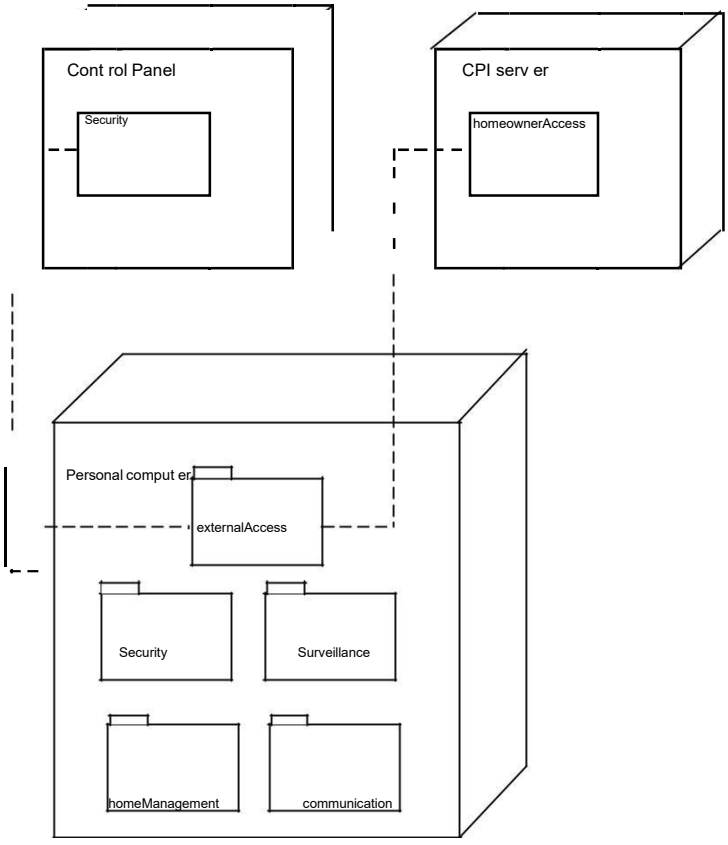
Fig: - UML int erfac e represent at ion for **Control Panel**

Component-level design elements: The component-level design for software is equivalent to a set of detailed drawings.

The component-level design for software fully describes the internal detail of each software component. To accomplish this, the component-level design defines data structures for all local data objects and algorithmic detail for all processing that occurs within a component and an interface that allows access to all component operations.



Deployment-level design elements: Deployment-level design elements indicated how software functionality and subsystems will be allocated within the physical computing environment that will support the software



ARCHITECTURAL DESIGN

SOFTWARE ARCHITECTURE: What Is Architecture?

Architectural design represents the structure of data and program components that are required to build a computer-based system. It considers the architectural style that the system will take, the structure and properties of the components that constitute the system, and the interrelationships that occur among all architectural components of a system.

The architecture is a representation that enables a software engineer to analyze the effectiveness of the design in meeting its stated requirements, consider architectural alternatives at a stage when making design changes is still relatively easy, (3) reducing the risks associated with the construction of the software.

The design of software architecture considers two levels of the design pyramid

i) data design

ii) architectural design.

Data design enables us to represent the data component of the architecture.

Architectural design focuses on the representation of the structure of software components, their properties, and interactions.

Why Is Architecture Important?

Bass and his colleagues [BAS98] identify three key reasons that software architecture is important:

- Representations of software architecture are an enabler for communication between all parties (stakeholders) interested in the development of a computer-based system.
- The architecture highlights early design decisions that will have a profound impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity.
- Architecture —constitutes a relatively small, intellectually graspable model of how the

system is structured and how its components work together

DATA DESIGN:

The data design activity translates data objects as part of the analysis model into data structures at the software component level and, when necessary, a database architecture at the application level.

At the program component level, the design of data structures and the associated algorithms required to manipulate them is essential to the creation of high-quality applications.

At the application level, the translation of a data model (derived as part of requirements engineering) into a database is pivotal to achieving the business objectives of a system.

At the business level, the collection of information stored in disparate databases and reorganized into a ―data warehouse‖ enables data mining or knowledge discovery that can have an impact on the success of the business itself.

Data design at the Architectural Level:

The challenge for a business has been to extract useful information from this data environment, particularly when the information desired is cross functional.

To solve this challenge, the business IT community has developed ***data mining*** techniques, also called ***knowledge discovery in databases*** (KDD), that navigate through existing databases in an attempt to extract appropriate business-level information. An alternative solution, called a ***data warehouse,*** adds an additional layer to the data architecture. a data warehouse is a large, independent database that encompasses some, but not all, of the data that are stored in databases that serve the set of applications required by a business.

Data design at the Component Level:

Data design at the component level focuses on the representation of data structures that are directly accessed by one or more software components. The following set of principles for data specification:

The systematic analysis principles applied to function and behavior should also be applied to data. All data structures and the operations to be performed on each should be identified.

A data dictionary should be established and used to define both data and program design. Low-level data design decisions should be deferred until late in the design process.

The representation of data structure should be known only to those modules that must make direct use of the data contained within the structure.

A library of useful data structures and the operations that may be applied to them should be developed.

A software design and programming language should support the specification and realization of abstract data types.

ARCHITECTURAL STYLES AND PATTERNS:

The builder has used an ***architectural style*** as a descriptive mechanism to differentiate the house from other styles (e.g., A-frame, raised ranch, Cape Cod).

The software that is built for computer-based systems also exhibits one of many architectural styles.

Each style describes a system category that encompasses

A set of ***components*** (e.g., a database, computational modules) that perform a function required by a system;

A set of ***connectors*** that enable ―communication, coordinations and cooperation‖ among components;

Constraints that define how components can be integrated to form the system; and

(4) ***Semantic models*** that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

An ***architectural pattern,*** like an architectural style, imposes a transformation the design of architecture. However, a pattern differs from a style in a number of fundamental ways:

The scope of a pattern is less broad, focusing on one aspect of the architecture rather than the architecture in its entirety.

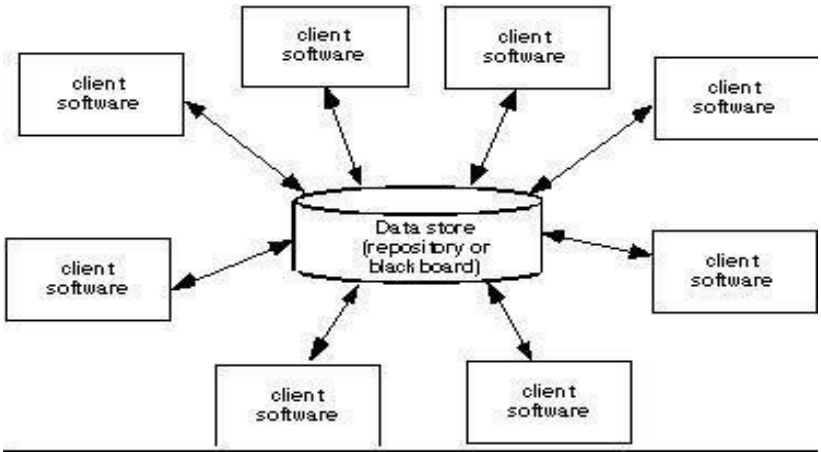
A pattern imposes a rule on the architecture, describing how the software will handle some aspect of its functionality at the infrastructure level.

Architectural patterns tend to address specific behavioral issues within the context of the architectural.

A Brief Taxonomy of Styles and Patterns Data-centered architectures:

A data store (e.g., a file or database) resides at the center of this architecture and is accessed frequently by other components that update, add, delete, or otherwise modify data within the store. A variation on this approach transforms the repository into a ―blackboard‖ that sends notification to client software when data of interest to the client changes

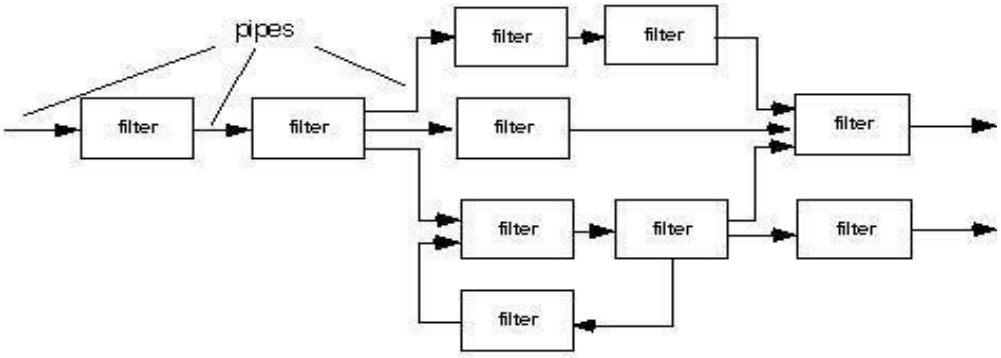
Data-centered architectures promote ***integrability***. That is, existing components can be changed



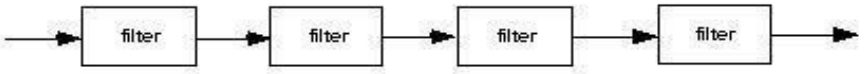
and new client components can be added to the architecture without concern about other clients (because the client components operate independently). In addition, data can be passed among clients using the blackboard mechanism

Data-flow architectures. This architecture is applied when input data are to be transformed through a series of computational or manipulative components into output data. A ***pipe and filter pattern*** has a set of components, called ***filters,*** connected by pipes that transmit data from one component to the next. Each filter works independently of those components upstream and downstream, is designed to expect data input of a certain form, and produces data output of a specified form.

If the data flow degenerates into a single line of transforms, it is termed ***batch sequential***. This pattern accepts a batch of data and then applies a series of sequential components (filters) to transform it.



(a) pipes and filters



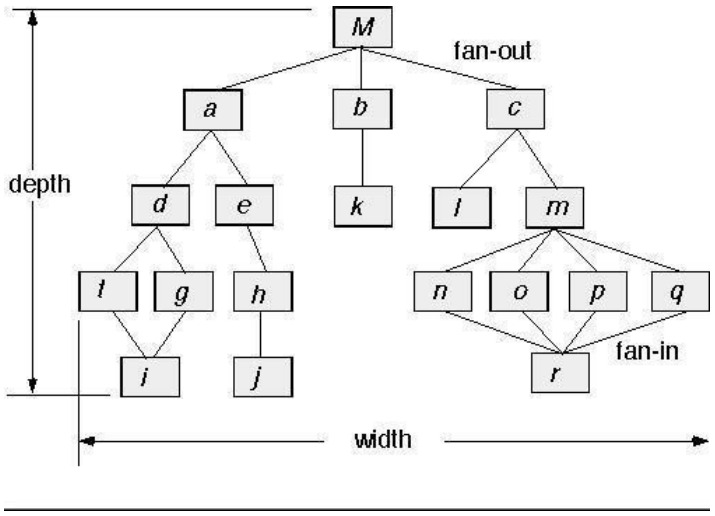
(b) batch sequential

Call and return architectures. This architectural style enables a software designer (system architect) to achieve a program structure that is relatively easy to modify and scale. A number of substyles [BAS98] exist within this category:

Main program/subprogram architectures. This classic program structure decomposes function into a control hierarchy where a ―main‖ program invokes a number of program components, which in turn may invoke still other components. Figure 13.3 illustrates an architecture of this

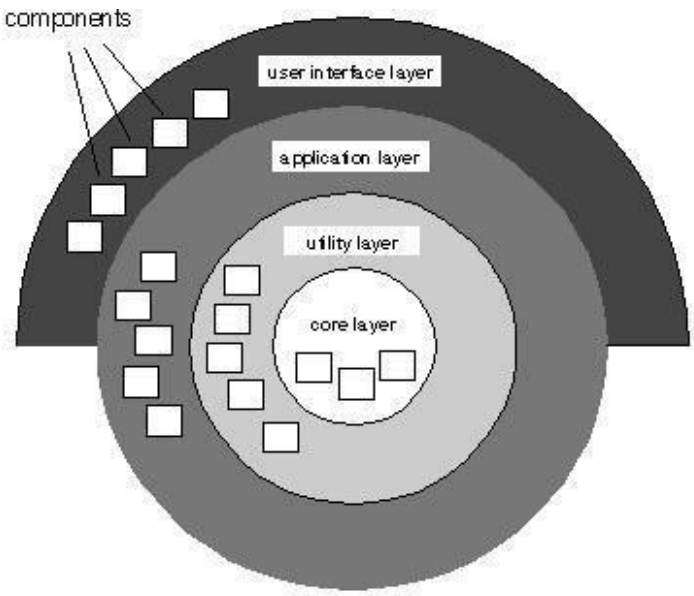
type.

Remote procedure call architectures. The components of a main program/ subprogram architecture are distributed across multiple computers on a network.



Object-oriented architectures. The components of a system encapsulate data and the operations that must be applied to manipulate the data. Communication and coordination between components is accomplished via message passing.

Layered architectures. The basic structure of a layered architecture is illustrated in Figure 14.3. A number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set. At the outer layer, components service user interface operations. At the inner layer, components perform operating system interfacing. Intermediate layers provide utility services and application software functions.



Architectural Patterns:

An ***architectural pattern,*** like an architectural style, imposes a transformation the design of architecture. However, a pattern differs from a style in a number of fundamental ways:

The scope of a pattern is less broad, focusing on one aspect of the architecture rather than the architecture in its entirety.

A pattern imposes a rule on the architecture, describing how the software will handle some aspect of its functionality at the infrastructure level.

Architectural patterns tend to address specific behavioral issues within the context of the architectural.

The architectural patterns for software define a specific approach for handling some behavioral characteristics of the system

Concurrency—applications must handle multiple tasks in a manner that simulates parallelism

- *operating system process management* pattern
- *task scheduler* pattern

Persistence—Data persists if it survives past the execution of the process that created it. Two patterns are common:

A ***database management system*** pattern that applies the storage and retrieval capability of a DBMS to the application architecture

An ***application level persistence*** pattern that builds persistence features into the application architecture

Distribution— the manner in which systems or components within systems communicate with one another in a distributed environment

A ***broker*** acts as a ‗middle-man' between the client component and a server component**.**

Organization and Refinement:

The design process often leaves a software engineer with a number of architectural alternatives, it is important to establish a set of design criteria that can be used to assess an architectural design that is derived. The following questions provide insight into the architectural style that has been derived:

Control.

How is control managed within the architecture?

Does a distinct control hierarchy exist, and if so, what is the role of components within this control hierarchy?

How do components transfer control within the system?

How is control shared among components?

Data.

How are data communicated between components?

Is the flow of data continuous, or are data objects passed to the system sporadically?

What is the mode of data transfer (i.e., are data passed from one component to another or are data available globally to be shared among system components)?

Do data components (e.g., a blackboard or repository) exist, and if so, what is their role? How do functional components interact with data components?

Are data components *passive* or *active* (i.e., does the data component actively interact with other components in the system)? How do data and control interact within the system?

4) ARCHITECTURAL DESIGN:

Representing the System in Context:

At the architectural design level, a software architect uses an architectural context diagram (ACD) to model the manner in which software interacts with entities external to its boundaries. The generic structure of the architectural context diagram is illustrated in the figure

Superordinate systems – those systems that use the target system as part of some higher level processing scheme.

Subordinate systems - those systems that are used by the target system and provide data or processing that are necessary to complete target system functionality.

Peer-level systems - those systems that interact on a peer-to-peer basis

Actors -those entities that interact with the target system by producing or consuming information thatis necessary for requisite processing

Defining Archetypes:

An archetype is a class or pattern that represents a core abstraction that is critical to the design of architecture for the target system. In general, a relative small set of archetypes is required to design even relatively complex systems.

In many cases, archetypes can be derived by examining the analysis classes defined as part of the analysis model. In safe home security function, the following are the archetypes:

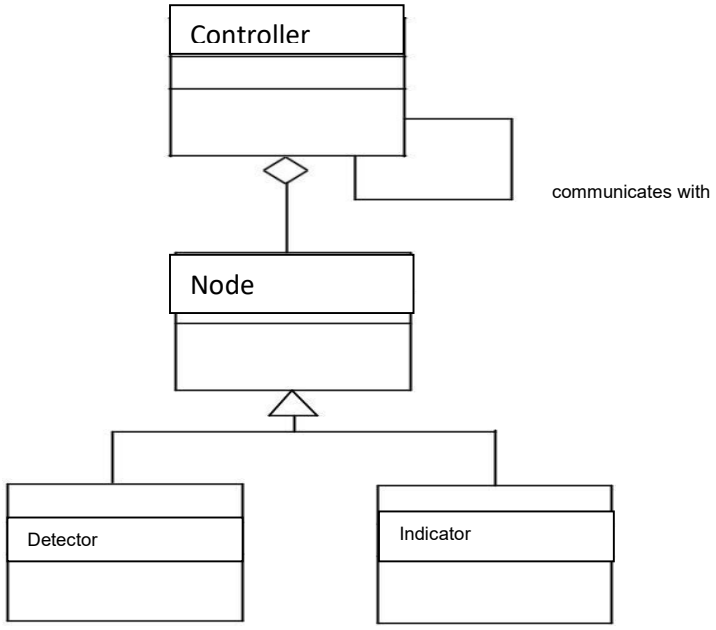
Node: Represent a cohesive collection of input and output elements of the home security function. For example a node might be comprised of (1) various sensors, and (2) a variety of alarm indicators.

Detector: An abstraction that encompasses all sensing equipment that feeds information into the target system

Indicator: An abstraction that represents all mechanisms for indication that an alarm condition is occurring.

Controller: An abstraction that depicts the mechanism that allows the arming or disarming of a node. If controllers reside on a network, they have the ability to communicate with one another.

Fig: - UML relat ionships for SafeHome security function archetypes



Refining the Architecture into Components:

As the architecture is refined into components, the structure of the system begins to emerge. The architectural designer begins with the classes that were

described as part of the analysis model. These analysis classes represent entities within the application domain that must be addressed within the software architecture. Hence, the application domain is one source is the infrastructure domain. The architecture must accommodate many infrastructure components that enable application domain. *For eg:* memory management components, communication components database components, and task management components are often integrated into the software architecture.

In the *safeHome* security function example, we might define the set of top-level components that address the following functionality:

External communication management- coordinates communication of the security function with external entities

Control panel processing- manages all control panel functionality.

Detector management- coordinates access to all detectors attached to the system.

Alarm processing- verifies and acts on all alarm conditions.

Design classes would be defined for each. It is important to note, however, that the design details of all attributes and operations would not be specified until component-level design.

Describing Instantiations of the System: An actual instantiation of the architecture means the architecture is applied to a specific problem with the intent of demonstrating that the structure and components are appropriate.

Object And Object Classes

Object : An object is an entity that has a state and a defined set of operations that operate on that state.

An obect class defination is both a type specification and a template for creating obects.

It includes declaration of all the attributes and operations that are associated with object of that class.

Object Oriented Design Process

There are five stages of object oriented design process

1)Understand and define the context and the modes of use of the system.

2)Design the system architecture

3)Identify the principle objects in the system.

4)Develop a design models

5) Specify the object interfaces Systems context and modes of use

It specify the context of the system.it also specify the relationships between the software that is being designed and its external environment.

If the system context is a static model it describe the other system in that environment.

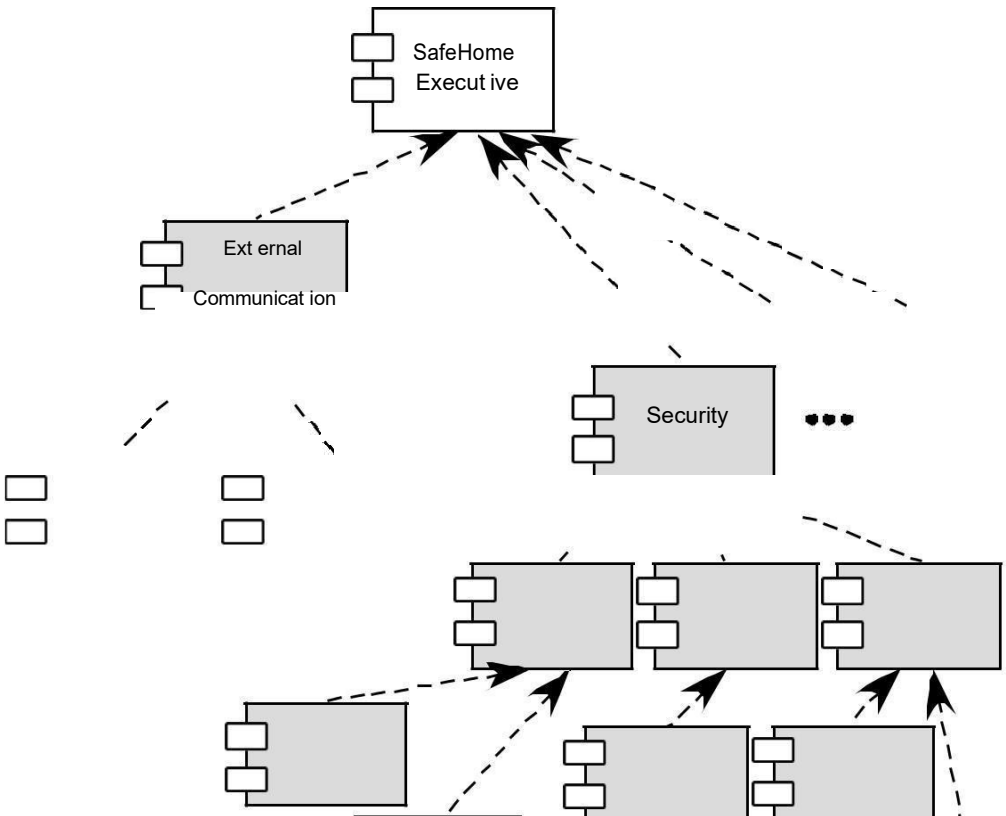
If the system context is a dynamic model then it describe how the system actually interact with the environment.

System Architecture

Once the interaction between the software system that being designed and the system environment have been defined

We can use the above information as basis for designing the System Architecture.

Object Identification



This process is actually concerned with identifying the object classes. We can identify the object classes by the following

1)Use a grammatical analysis 2)Use a tangible entities 3)Use a behaviourial approach

4) Use a scenario based approach Design model

Design models are the bridge between the requirements and implementation. There are two type of design models

1) Static model describe the relationship between the objects.

2)Dynamic model describe the interaction between the objects

Object Interface SpecificationIt is concerned with specifying the details of the interfaces to an objects.

Design evolution

The main advantage OOD approach is to simplify the problem of making changes to the design. Changing the internal details of an obect is unlikely to affect any other system object.

Unit-4

Introduction to Testing

- Testing is a set of activities which are decided in advance i.e before the start of development and organized systematically.
- In the literature of software engineering various testing strategies to implement the testing are defined.
- All the strategies give a testing template.

Following are the characteristic that process the testing templates:

- The developer should conduct the successful technical reviews to perform the testing successful.
- Testing starts with the component level and work from outside toward the integration of the whole computer based system.
- Different testing techniques are suitable at different point in time.
- Testing is organized by the developer of the software and by an independent test group.
- Debugging and testing are different activities, then also the debugging should be accommodated in any strategy of testing.

Difference between Verification and Validation

Verification	Validation
Verification is the process to find whether the software meets the specified requirements for particular phase.	The validation process is checked whether the software meets requirements and expectation of the customer.
It estimates an intermediate product.	It estimates the final product.
The objectives of verification is to check whether software is constructed according to requirement and design specification.	The objectives of the validation is to check whether the specifications are correct and satisfy the business need.
It describes whether the outputs are as per the inputs or not.	It explains whether they are accepted by the user or not.
Verification is done before the validation.	It is done after the verification.
Plans, requirement, specification, code are evaluated during the verifications.	Actual product or software is tested under validation.
It manually checks the files and document.	It is a computer software or developed program based checking of files and document.

Strategy of testing

A strategy of software testing is shown in the context of spiral.

Following figure shows the testing strategy:

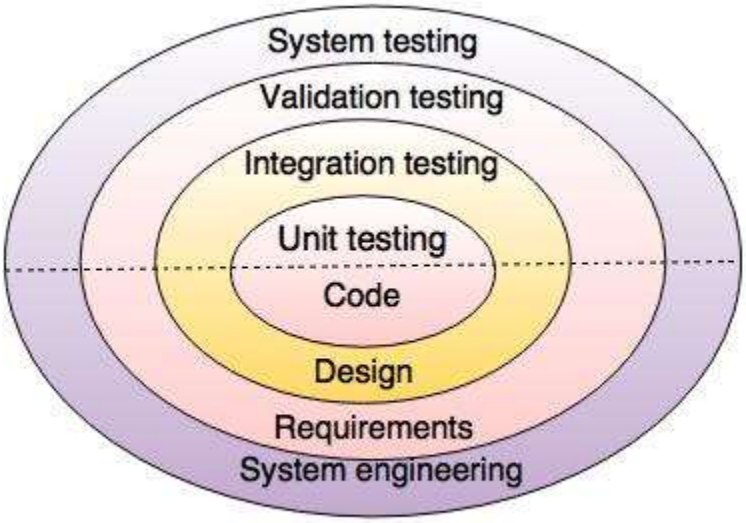


Fig. - Testing Strategy

Unit testing
Unit testing starts at the centre and each unit is implemented in source code.

Integration testing
An integration testing focuses on the construction and design of the software.

Validation testing
Check all the requirements like functional, behavioral and performance requirement are validate against the construction software.

System testing
System testing confirms all system elements and performance are tested entirely.

Testing strategy for procedural point of view

As per the procedural point of view the testing includes following steps.

1) Unit testing
2) Integration testing
3) High-order tests
4) Validation testing

These steps are shown in following figure:

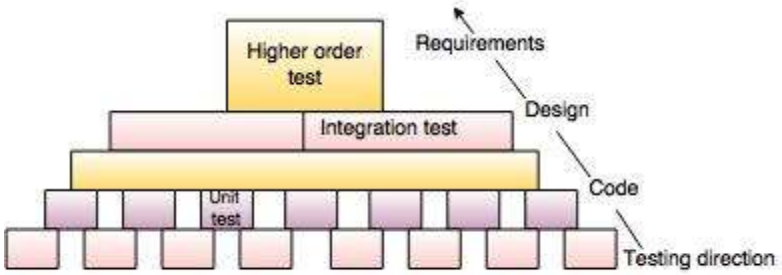


Fig.- Steps of software testing

Following are the issues considered to implement software testing strategies.

- Specify the requirement before testing starts in a quantifiable manner.
- According to the categories of the user generate profiles for each category of user.
- Produce a robust software and it's designed to test itself.
- Should use the Formal Technical Reviews (FTR) for the effective testing.
- To access the test strategy and test cases FTR should be conducted.
- To improve the quality level of testing generate test plans from the users feedback.

Test strategies for conventional software

Following are the four strategies for conventional software:
1) Unit testing
2) Integration testing
3) Regression testing
4) Smoke testing

1) Unit testing

- Unit testing focus on the smallest unit of software design, i.e module or software component.
- Test strategy conducted on each module interface to access the flow of input and output.
- The local data structure is accessible to verify integrity during execution.
- Boundary conditions are tested.
- In which all error handling paths are tested.
- An Independent path is tested.

Following figure shows the unit testing:

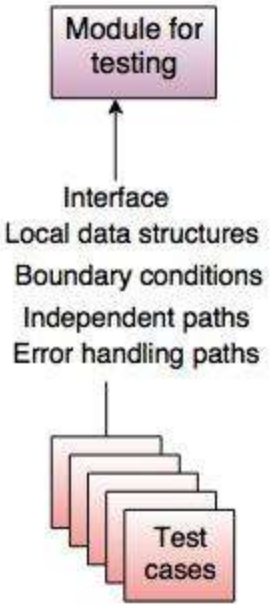


Fig. - Unit test

Unit test environment

The unit test environment is as shown in following figure:

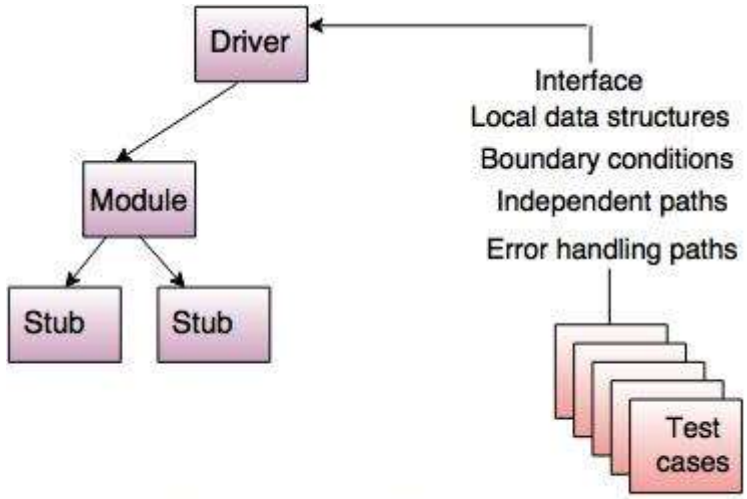


Fig. - Unit test environment

Difference between stub and driver

Stub	Driver
Stub is considered as subprogram.	It is a simple main program.
Stub does not accept test case data.	Driver accepts test case data.
It replace the modules of the program into subprograms and are tested by the next driver.	Pass the data to the tested components and print the returned result.

2) Integration testing

Integration testing is used for the construction of software architecture.

There are two approaches of incremental testing are:
i) Non incremental integration testing
ii) Incremental integration testing

i) Non incremental integration testing

- Combines all the components in advanced.
- A set of error is occurred then the correction is difficult because isolation cause is complex.

ii) Incremental integration testing

- The programs are built and tested in small increments.
- The errors are easier to correct and isolate.
- Interfaces are fully tested and applied for a systematic test approach to it.

Following are the incremental integration strategies:
a. Top-down integration
b. Bottom-up integration

a. Top-down integration

- It is an incremental approach for building the software architecture.
- It starts with the main control module or program.
- Modules are merged by moving downward through the control hierarchy.

Following figure shows the top down integration.

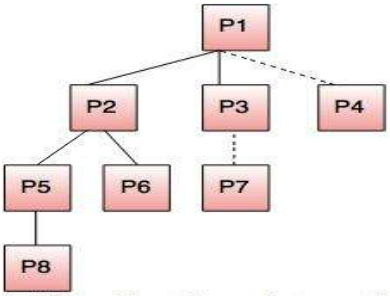


Fig. - Top-down integration

Problems with top-down approach of testing

Following are the problems associated with top-down approach of testing as follows:

- Top-down approach is an incremental integration testing approach in which the test conditions are difficult to create.
- A set of errors occur, then correction is difficult to make due to the isolation of cause.
- The programs are expanded into various modules due to the complications.
- If the previous errors are corrected, then new get created and the process continues. This situation is like an infinite loop.

b. Bottom-up integration

In bottom up integration testing the components are combined from the lowest level in the program structure.

The bottom-up integration is implemented in following steps:

- The low level components are merged into clusters which perform a specific software sub function.
- A control program for testing(driver) coordinate test case input and output.
- After these steps are tested in cluster.

- The driver is removed and clusters are merged by moving upward on the program structure.

Following figure shows the bottom up integration:

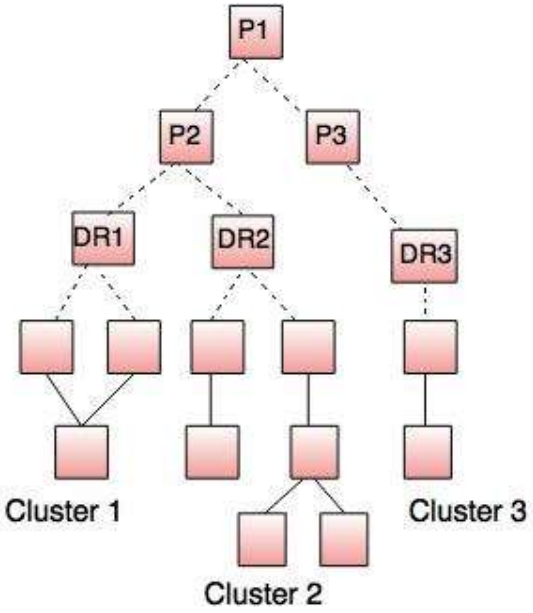


Fig. - Bottom-up integration

3) Regression testing

- In regression testing the software architecture changes every time when a new module is added as part of integration testing.

4) smoke testing

- The developed software component are translated into code and merge to complete the product.

Difference between Regression and smoke testing

Regression testing	Smoke testing
Regression testing is used to check defects generated to other modules by making the changes in existing programs.	At the time of developing a software product smoke testing is used.
In regression tested components are tested again to verify the errors.	It permit the software development team to test projects on a regular basis.

Regression testing needs extra manpower because the cost of the project increases.	Smoke testing does not need an extra manpower because it does not affect the cost of project.
Testers conduct the regression testing.	Developer conducts smoke testing just before releasing the product.

Alpha and Beta testing

Alpha testing	Beta testing
Alpha testing is executed at developers end by the customer.	Beta testing is executed at end-user sites in the absence of a developer.
It handles the software project and applications.	It usually handles software product.
It is not open to market and the public.	It is always open to the market and the public.
Alpha testing does not have any different name.	Beta testing is also known as the field testing.
Alpha testing is not able to test the errors because the developer does not known the type of user.	In beta testing, the developer corrects the errors as users report the problems.
In alpha testing, developer modifies the codes before release the software without user feedback.	In beta testing, developer modifies the code after getting the feedback from user.

System testing

- System testing is known as the testing behavior of the system or software according to the software requirement specification.
- It is a series of various tests.
- It allows to test, verify and validate the business requirement and application architecture.
- The primary motive of the tests is entirely to test the computer-based system.

Following are the system tests for software-based system

1. Recovery testing

- To check the recovery of the software, force the software to fail in various ways.
- Reinitialization, check pointing mechanism, data recovery and restart are evaluated correctness.

2. Security testing

- It checks the system protection mechanism and secure improper penetration.

3. Stress testing

- System executes in a way which demands resources in abnormal quantity, frequency.
- A variation of stress testing is known as sensitivity testing.

4. Performance testing

- Performance testing is designed to test run-time performance of the system in the context of an integrated system.
- It always combines with the stress testing and needs both hardware and software requirements.

5. Deployment testing

- It is also known as configuration testing.
- The software works in each environment in which it is to be operated.

Debugging process

- Debugging process is not a testing process, but it is the result of testing.
- This process starts with the test cases.
- The debugging process gives two results, i.e the cause is found and corrected second is the cause is not found.

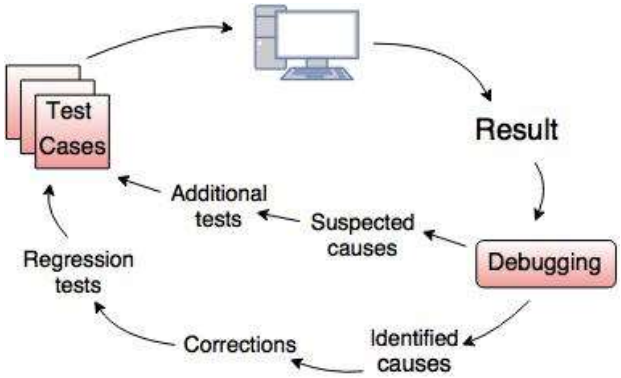


Fig. - Debugging process

Debugging Strategies

Debugging identifies the correct cause of error.

Following are the debugging strategies:

1. Brute force

- Brute force is commonly used and least efficient method for separating the cause of software error.
- This method is applied when all else fails.

2. Backtracking

- Backtracking is successfully used in small programs.
- The source code is traced manually till the cause is found.

3. Cause elimination

- Cause elimination establishes the concept of binary partitioning.
- It indicates the use of induction or deduction.
- The data related to the error occurrence is arranged in separate potential cause.

Characteristics of testability

Following are the characteristics of testability:

1. Operability

- If a better quality system is designed and implemented then it easier to test.

2. Observability

- It is an ability to see which type of data is being tested.
- Using observability it will easily identify the incorrect output.
- Catch and report the internal errors automatically.

3. Controllability

- If the users controlled the software properly then the testing is automated and optimized better.

4. Decomposability

- The software system is constructed from independent module then tested independently.

5. Simplicity

- The programs must display the functional, structural, code simplicity so that programs are easier to test.

6. Stability

- Changes are rare during the testing and do not disprove existing tests.

7. Understandability

- The architectural designs are well understood.
- The technical documentation is quickly accessible, organized and accurate.

Attributes of 'good' test

- The possibility of finding an error is high in good test.
- Limited testing time and resources. There is no purpose to manage same test as another test.
- A test should be used for the highest probability of uncovering the errors of a complete class.
- The test must be executed separately and it should not be too simple nor too complex.

Difference between white and black box testing

White-Box Testing	Black-box Testing
White-box testing known as glass-box testing.	Black-box testing also called as behavioral testing.

It starts early in the testing process.	It is applied in the final stages of testing.
In this testing knowledge of implementation is needed.	In this testing knowledge of implementation is not needed.
White box testing is mainly done by the developer.	This testing is done by the testers.
In this testing, the tester must be technically sound.	In black box testing, testers may or may not be technically sound.
Various white box testing methods are: Basic Path Testing and Control Structure Testing.	**Various black box testing are:** Graph-Based testing method, Equivalence partitioning, Boundary Value Analysis, Orthogonal Array Testing.

Basic Path Testing

- Basic path testing is proposed by Tom McCabe.
- It is a white box testing technique.
- It allows the test case designer to obtain a logical complexity measure of a procedural design.
- This measure guides for defining a basic set of execution paths.

The examples of basic path testing are as follows:

- Flow graph notation
- Independent program paths
- Deriving test cases
- Graph matrices

Control structure testing

The control structure testing is a wide testing study and also improves the quality of white-box testing.

The examples of white-box testing is as follows:

1) Conditional testing
All the program module and the logical conditions in the program are tested in conditional testing.

2) Data flow testing
A test path of a program is selected as per the location of definitions and the uses of variables in the program.

3) Loop testing

It is a white-box testing technique. Loop testing mainly focuses on the validity of the loop constructs.

Four different classes of loops are:

1. Simple loops
2. Concatenated loops
3. Nested loops
4. Unstructured loops

KOMMURI PRATAP REDDY INSTITUTE OF TECHNOLOGY

Teaching Methods Implemented for Effective Teaching Learning Process

Academic Year: 2021-22	Branch: CSE	Year/ Semester: III/I
Name of the Faculty: A. Sai Kumar	Name of the Subject: Software Engineering	

Teaching Method: PPT & Classroom Discussion

Idea:

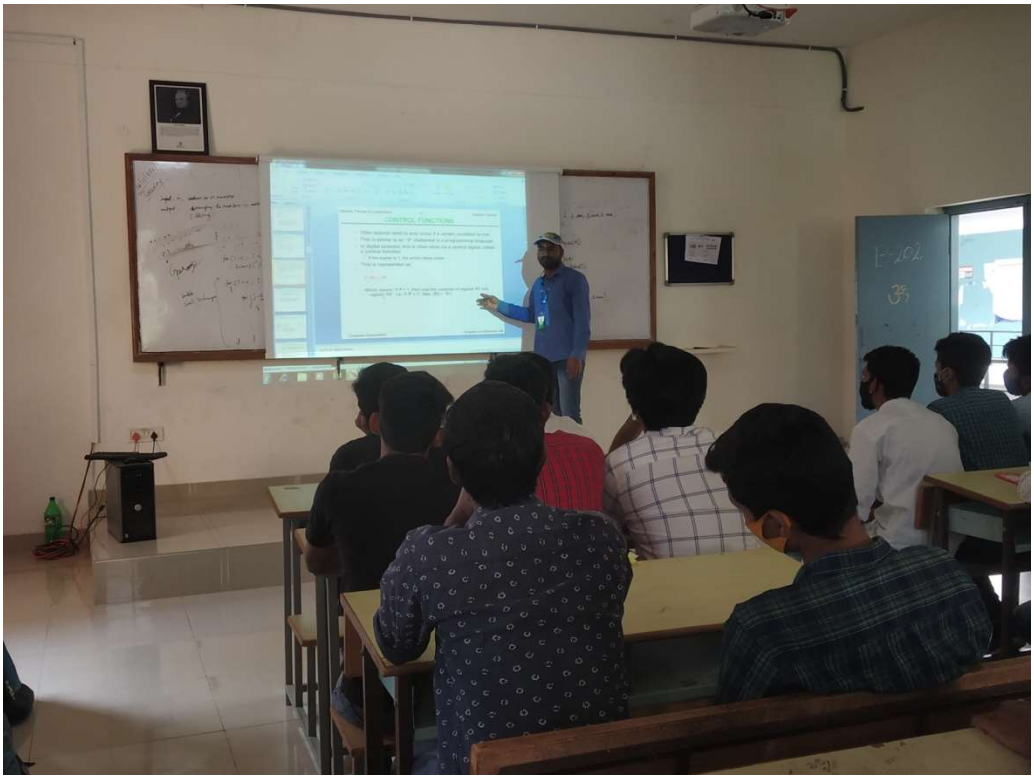
To make students discuss a given topic.

Implementation:

- PPT Presentation
- Slides Prepared for the selected topics in subject
- Diagrammatical explanation given with the PPT
- Select topic
- Students are divided into three groups.
- Each group is assigned a name based on topic selected.
- Students are asked to give their views on the concept.
- They are asked to prepare and give seminars on the given topics.

Outcome:

- Active participation of students.
- Remembering the topic for a longer time.


CONTROL FUNCTIONS

KOMMURI PRATAP REDDY INSTITUTE OF TECHNOLOGY

(COLLEGE OF ENGINEERING)

Ghanpuram(V), Ghatkesar (M), Hyderabad. - 501301

III B.Tech - I Sem(R18)	**DESCRIPTIVE TEST – I**	**A.Y.: 2021 - 2022**
Branch: CSE		Subject Name: Software Engineering
Class: **B.Tech**		Date of Exam : 08-11-2021
Max. Marks:10M		Time: 1:40 PM to 3:00 PM

Answer any **TWO** of the following questions: **2x 5 = 10M**

1. What is software? Explain about changing nature of software. **(CO1,Remember, Understand)**
2. Discuss about Functional and Non-Functional requirements. **(CO2, Understand)**
3. Describe the following process models
   a) Waterfall model b) Unified process model. **(CO1, Understand)**
4. What is feasibility study? Explain about it. **(CO2, Remember, Understand)**

KOMMURI PRATAP REDDY INSTITUTE OF TECHNOLOGY

(COLLEGE OF ENGINEERING)

Ghanpuram(V), Ghatkesar (M), Hyderabad -501301

IV B.Tech - I Sem(R18) **DESCRIPTIVE TEST – I** **A.Y.: 2021 - 2022**

Branch: CSE Engineering

Subject Name: Software

Class: **B.Tech**

Date of Exam : 08-11-2021

Max. Marks:10M

Time: 1:40 PM to 3:00 PM

Answer any **TWO** of the following questions: **2x 5 = 10**M

1. What is software? Explain about changing nature of software. **(CO1,Remember, Understand)**
2. Discuss about Functional and Non-Functional requirements. **(CO2, Understand)**

3. Describe the following process models
   a) Waterfall model b) Unified process model. **(CO1, Understand)**

		R	A		A	0			

4. What is feasibility study? Explain about it. **(CO2, Remember, Understand)**

KOMMURI PRATAP REDDY INSTITUTE OF TECHNOLOGY-GHATKESAR

III B.Tech I Sem. I Mid-Term Examinations, November-2021

Subject Name: Software Engineering **Branch: CSE**

Objective Exam

Name: _____Hall Ticket No.-

Answer All Questions. All Questions Carry Equal Marks.

Time: 20 Min. **Marks: 10.**

I. Choose the correct alternative:

1. What is a Software? ( )

A. Software is documentation and configuration of data
B. Software is set of programs
C. Software is set of programs, documentation & configuration of data
D. None of the mentioned

2. Which of these software engineering activities are not a part of software processes? ( )

A. Software development
B. Software dependence
C. Software validation
D. Software specification

3. Which one of the following models is not suitable for accommodating any change? ( )

A. Prototyping Model
B. RAD model
C. Build & fixe model
D. Waterfall model

4. The spiral model was originally proposed by ( )

A. Barry Boehm
B. Pressman
C. Royce
D. None

5. If you were to create client/server applications, which model would you go for? ( )

A. Concurrent Model
B. Spiral Model
C. WINWIN Spiral Model
D. Incremental Model

6. How many numbers of maturity levels in CMM are available? ( )

A. 6
B. 5
C. 4
D. 3

7. Which is not one of the types of prototype of Prototyping Model? ( )

A. Horizontal Prototype
B. Vertical Prototype
C. Diagonal Prototype
D. Domain Prototype

8. RAD stands for ( )

A. Relative Application Development
B. Rapid Application Development
C. Rapid Application Document
D. None of the mentioned

9. SDLC stands for ( )

A. Software Development Life Cycle
B. System Development Life cycle
C. Software Design Life Cycle
D. System Design Life Cycle

10. What are the characteristics of software ( )

A. Software is developed or engineered; it is not manufactured in the classical sense
B. Software does not “wear out”
C. Software can be custom built or custom build
D. All mentioned above

II Fill in the Blanks:

1. _____is the Application of science, tools and methods to find cost effective solution to problems.
2. CMMI stands for _____.
3. _____. defined as systematic, disciplined and quantifiable approach for the development, operation and maintenance of software
4. Give examples for system software _____.
5. Examples for application software _____.
6. MATLAB, AUTOCAD, PSPICE, ORCAD these are belongs to what kind of software _____.
7. _____ software used in instrumentation and control applications, washing machines, satellites, microwaves
8. Software development is totally a _____ technology.
9. Examples of artificial intelligence _____.
10. The software process framework is a collection of _____.

-oOo-

Software Engineering Mid-1 Key Paper

1.Q.A)

The term is made of two words, software and engineering.

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

Changing Nature of Software

Nowadays, seven broad categories of computer software present continuing challenges for software engineers .which is given below:

8. **System Software:**
   System software is a collection of programs which are written to service other programs. Some system software processes complex but determinate, information structures. Other system application process largely indeterminate data. Sometimes when, the system software area is characterized by the heavy interaction with computer hardware that requires scheduling, resource sharing, and sophisticated process management.
   Ex: - Examples of system software include operating systems like macOS, GNU/Linux , Android and Microsoft Windows, computational science software, game engines, industrial automation, and software as a service applications.
9. **Application Software:**
   Application software is defined as programs that solve a specific business need. Application in this area process business or technical data in a way that facilitates business operation or management technical decision making. In addition to convention data processing application, application software is used to control business function in real time.
   Ex:-

- Microsoft suite of products (Office, Excel, Word, PowerPoint, Outlook, etc.)
- Internet browsers like Firefox, Safari, and Chrome.

- Mobile pieces of **software** such as Pandora (for music appreciation), Skype (for real-time online communication), and Slack (for team collaboration)
- **Application software** (app for short) is a program or group of programs designed for end users. **Examples** of an **application** include a word processor, a spreadsheet, an accounting **application**, a web browser, an email client, a media player, a file viewer, simulators, a console game or a photo editor.

10. **Engineering and Scientific Software:**

This software is used to facilitate the engineering function and task. However modern application within the engineering and scientific area are moving away from the conventional numerical algorithms. Computer-aided design, system simulation, and other interactive applications have begun to take a real-time and even system software characteristic.

Ex: - Examples are **software** like MATLAB, AUTOCAD, PSPICE, ORCAD, etc.

11. **Embedded Software:**

Embedded software resides within the system or product and is used to implement and control feature and function for the end-user and for the system itself. Embedded software can perform the limited and esoteric function or provided significant function and control capability.

This type of software is embedded into the hardware normally in the Read Only Memory (ROM) as a part of a large system and is used to support certain functionality under the control conditions.

Examples are software used in instrumentation and control applications, washing machines, satellites, microwaves, washing machines etc.

12. **Product-line Software:**

Designed to provide a specific capability for use by many different customers, product line software can focus on the limited and esoteric marketplace or address the mass consumer market.

13. **Web Application:**

It is a client-server computer program which the client runs on the web browser. In their simplest form, Web apps can be little more than a set of linked hypertext files that present information using text and limited graphics. However, as e-commerce and B2B application grow in importance. Web apps are evolving into a sophisticate computing environment that not only provides a standalone feature, computing function, and content to the end user.

Web applications include online forms, shopping carts, word processors, spreadsheets, video and photo editing, file conversion, file scanning, and email programs such as Gmail, Yahoo and AOL. Popular **applications** include Google **Apps** and Microsoft 365.Online retail sales, online auctions, wikis, instant messaging services and more.

14. **Artificial Intelligence Software:**

Artificial intelligence software makes use of a nonnumerical algorithm to solve a complex problem that is not amenable to computation or straightforward analysis. Application within this area includes robotics, expert system, pattern recognition, artificial neural network, theorem proving and game playing.

Example, speech recognition, problem-solving, learning and planning.

2.Q.A)

Functional Requirements:-

Requirements, which are related to functional aspect of software fall into this category.
They define functions and functionality within and from the software system.

Examples -

- ✓ Search option given to user to search from various invoices.
- ✓ User should be able to mail any report to management.
- ✓ Users can be divided into groups and groups can be given separate rights.
- ✓ Should comply business rules and administrative functions.
- ✓ Software is developed keeping downward compatibility intact.

Non-Functional Requirements:-

Requirements, which are not related to functional aspect of software, fall into this category. They are implicit or expected characteristics of software, which users make assumption of.

Non-functional requirements include -

- Security
- Logging
- Storage
- Configuration
- Performance
- Cost
- Interoperability
- Flexibility
- Disaster recovery
- Accessibility

Requirements are categorized logically as

- **Must Have** : Software cannot be said operational without them.
- **Should have** : Enhancing the functionality of software.
- **Could have** : Software can still properly function with these requirements.
- **Wish list** : These requirements do not map to any objectives of software.

While developing software, 'Must have' must be implemented, 'Should have' is a matter of debate with stakeholders and negation, whereas 'could have' and 'wish list' can be kept for software updates.

3.Q.A)

The waterfall model:-

- The waterfall model is also called as **'Linear sequential model'** or **'Classic life cycle model'.**
- In this model, each phase is fully completed before the beginning of the next phase.
- This model is used for the small projects.
- In this model, feedback is taken after each phase to ensure that the project is on the right path.
- Testing part starts only after the development is complete.

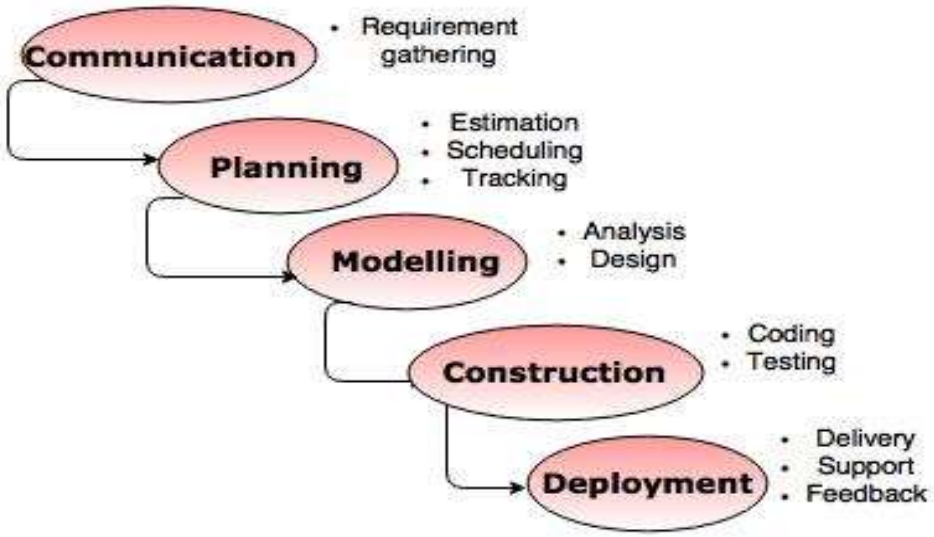


Fig. - The Waterfall model

NOTE: The description of the phases of the waterfall model is same as that of the process model.

An alternative design for 'linear sequential model' is as follows:

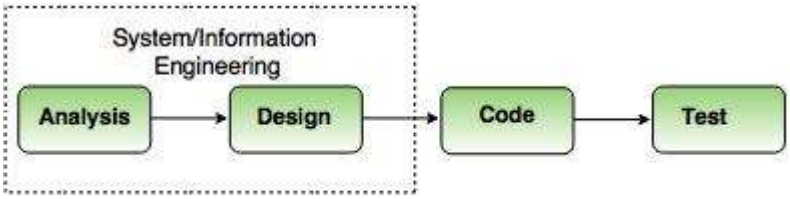


Fig. - The linear sequential model

Advantages of waterfall model:-

- The waterfall model is simple and easy to understand, implement, and use.
- All the requirements are known at the beginning of the project, hence it is easy to manage.
- It avoids overlapping of phases because each phase is completed at once.
- This model works for small projects because the requirements are understood very well.
- This model is preferred for those projects where the quality is more important as compared to the cost of the project.

Disadvantages of the waterfall model:-

- This model is not good for complex and object oriented projects.
- It is a poor model for long projects.
- The problems with this model are uncovered, until the software testing.
- The amount of risk is high.

The waterfall model:-

- The waterfall model is also called as **'Linear sequential model'** or **'Classic life cycle model'.**
- In this model, each phase is fully completed before the beginning of the next phase.
- This model is used for the small projects.
- In this model, feedback is taken after each phase to ensure that the project is on the right path.
- Testing part starts only after the development is complete.

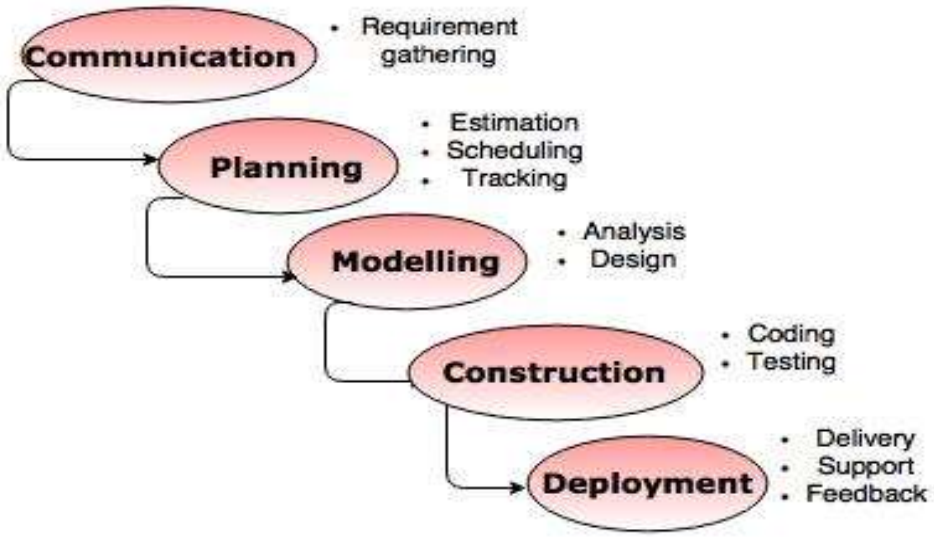


Fig. - The Waterfall model

NOTE: The description of the phases of the waterfall model is same as that of the process model.

An alternative design for 'linear sequential model' is as follows:

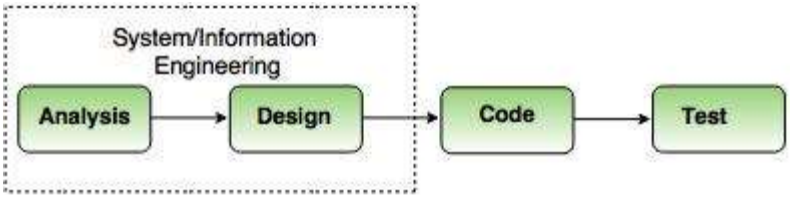


Fig. - The linear sequential model

Advantages of waterfall model:-

- The waterfall model is simple and easy to understand, implement, and use.
- All the requirements are known at the beginning of the project, hence it is easy to manage.
- It avoids overlapping of phases because each phase is completed at once.
- This model works for small projects because the requirements are understood very well.

- This model is preferred for those projects where the quality is more important as compared to the cost of the project.

Disadvantages of the waterfall model:-

- This model is not good for complex and object oriented projects.
- It is a poor model for long projects.
- The problems with this model are uncovered, until the software testing.
- The amount of risk is high.

PHASES OF THEUNIFIED PROCESS:

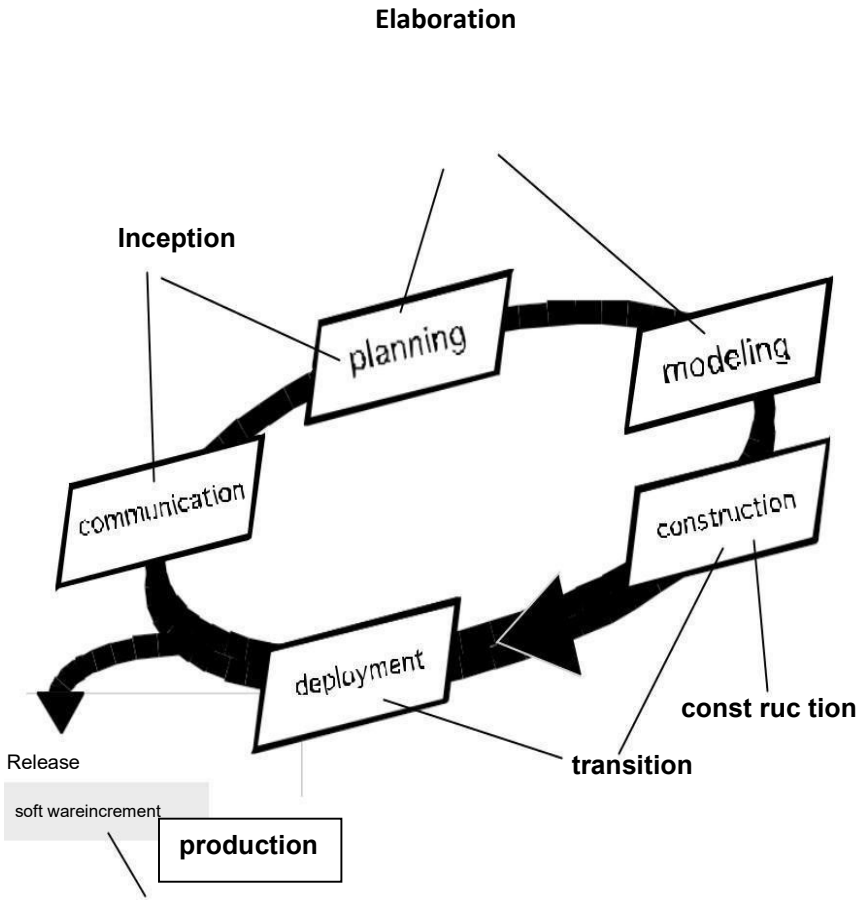
The **inception** phase of the UP encompasses both customer communication and planning activities. By collaborating with the customer and end-users, business requirements for the software are identified, a rough architecture for the system is proposed and a plan for the iterative, incremental nature of the ensuing project is developed.

The **elaboration** phase encompasses the customer communication and modeling activities of the generic process model. Elaboration refines and expands the preliminary use-cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software- the use-case model, the analysis model, the design model, the implementation model, and the deployment model.

The **construction** phase of the UP is identical to the construction activity defined for the generic software process. Using the architectural model as input, the construction phase develops or acquires the software components that will make each use-case operational for end-users. To accomplish this, analysis and design models that were started during the elaboration phase are completed to reflect the final version of the software increment.

The **transition** phase of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment activity. Software given to end-users for beta testing, and user feedback reports both defects and necessary changes.

The **production** phase of the UP coincides with the deployment activity of the generic process. During this phase, the on-going use of the software is monitored, support for the operating environment is provided, and defect reports and requests for changes are submitted and evaluated.



4.Q.A)

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:

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

Software Engineering Mid-1 Objective Key Paper

Choose the correct answer from the choices

1. C
2. A
3. D
4. A
5. A
6. B
7. C
8. B
9. A
10. D

Fill the blanks with correctanswer.

1. Engineering
2. Capability Maturity Model Integration
3. Software Engineering
4. OS linux, windows, game engines
5. ms office, internet browsers, mobile apps
6. engineering and scientific software
7. embedded
8. layered
9. speech recognition, robots, smart assistant, disease mapping
10. task sets

KOMMURI PRATAP REDDY INSTITUTE OF TECHNOLOGY

(COLLEGE OF ENGINEERING)

Ghanpuram(V), Ghatkesar (M), Hyderabad. - 501301

III B.Tech - I Sem(R18)	**DESCRIPTIVE TEST – II**	**A.Y.: 2021 - 2022**

Branch: CSE

Subject Name: Software Engineering

Class: **B.Tech**

Date of Exam: 03-02-2022

Max. Marks:10M

Time: 1:40 PM to 2:40 PM

Answer any **TWO** of the following questions: **2x 5 = 10M**

1. Explain about the test strategies for conventional software? (CO4, Understand)
2. Compare black box testing with white box testing? (CO4, Analyze)
3. Explain about software risks? (CO5, understand)
4. Explain about following software metrics (CO5, Understand)
   a) Process metrics b) Product metrics

KOMMURI PRATAP REDDY INSTITUTE OF TECHNOLOGY

(COLLEGE OF ENGINEERING)

Ghanpuram(V), Ghatkesar (M), Hyderabad -501301

III B.Tech - I Sem(R18) **DESCRIPTIVE TEST – II** **A.Y.: 2021 - 2022**

Branch: CSE Engineering

Subject Name: Software

Class: **B.Tech**

Date of Exam: 03-02-2022

Max. Marks:10M

Time: 1:40 PM to 2:40 PM

Answer any **TWO** of the following questions: **2x 5 = 10**M

1. Explain about the test strategies for conventional software? (CO4, Understand)
2. Compare black box testing with white box testing? (CO4, Analyze)
3. Explain about software risks? (CO5, Understand)

4. Explain about following software metrics (CO5, Understand)

   a) Process metrics b) Product metrics

		R	A		A	0			

Code No: CS502PC

KOMMURI PRATAP REDDY INSTITUTE OF TECHNOLOGY-GHATKESAR

III B.Tech I Sem. II Mid-Term Examinations, February-2022

Subject Name: SOFTWARE ENGINEERING **Branch: CSE**

Objective Exam

Name: _____Hall Ticket No.-

Answer All Questions. All Questions Carry Equal Marks.

Time: 20 Min. **Marks: 10.**

II. Choose the correct alternative:

1. The tools that support different stages of software development life cycle are called ( )

E. CASE Tools
F. CAME tools
G. CAQE tools
H. CARE tools

2. Alpha and Beta testing are forms of ( )

E. Acceptance testing
F. Integration testing
G. System testing
H. Unit testing

3. The most important feature of spiral model is ( )

E. Requirement analysis
F. Risk management
G. Quality management
H. Configuration management

4. One of the fault base testing technique is ( )

E. Unit testing
F. Beta testing
G. Stress testing
H. Mutation testing

5. SRS is also known as specification of ( )

E. White box testing
F. Integrated testing
G. Stress testing
H. Black box testing

6. In the spiral model 'risk analysis' is performed ( )

E. In the first loop
F. In the first and second loop
G. In every loop
H. Before using spiral model

7. Which of the following is an indirect measure of product? ( )

E. Quality
F. Complexity
G. Reliability

H. All of the mentioned

Cont.....2

8. In size oriented metrics, metrics are developed based on the ( )

E. Number of functions
F. Number of user inputs
G. Number of lines of code
H. Amount of memory usage

9. Black-box testing also called as ( )

E. logical testing
F. structural testing
G. behavioral testing
H. none of the above

10. Black box testing is done by the ( )

E. Testers
F. Developer
G. Users
H. All mentioned above

II Fill in the Blanks:

11. _____ design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them.
12. RMMM stands for _____.
13. _____ design is the specification of the interaction between a system and its environment.

14. _____ Testing is based on the application's internal code structure.
15. _____ is a software testing method in which testers evaluate the functionality of the software under test without looking at the internal code structure
16. An integration testing focuses on the _____of the software.
17. _____ is considered as subprogram.
18. _____ is a simple main program.
19. The debugging process gives two results they are _____.
20. White-box testing known as _____.

-oOo-

Software Engineering Mid-2 (Objective key)

Multiple Choice Questions

1) A

2) A

3) B

4) D

5) D

6) C

7) D

8) C

9) C

10) A

Fill in the Blanks

1) Architectural Design

2) Risk Mitigation, Monitoring and Management

3) *Interface design*

4) White box testing

5) Black box teting

6) construction and design

7) stub

8) driver

9) the cause is found and corrected second is the cause is not found

10) glass-box testing