MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu
Estd. 2000
IQAC

L-1

LECTURE HANDOUTS

CSE | **II/IV**

Course Name with Code : 19CSC12/Software Engineering
Course Teacher : V.Karuppuchamy
Unit : I- Software Process And Agile Development **Date of Lecture :** 22/2/2021

Topic of Lecture : Introduction to Software Engineering

Introduction : (Maximum 5 sentences) :

- Software is a set of instructions that provides desired features, function, and performance on execution.
- Software is a data structure that enable the programs to adequately manipulate information
- Software is developed or engineered and not manufactured.
- Software doesn't wear out.
- Although the industry is moving toward component-based construction, most software continues to be custom built.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Software Basics
- Physical components
- Types of Software
- Hardware Basics

Detailed content of the Lecture:

What is Software Engineering?

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

The **software** is a collection of integrated programs.

Software subsists of carefully-organized instructions and code written by developers on any of various particular computer languages.

Computer programs and related documentation such as requirements, design models and user manuals.

Engineering is the application of **scientific** and **practical** knowledge to **invent, design, build, maintain,** and **improve frameworks, processes, etc**.

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

Why is Software Engineering required?

Software Engineering is required due to the following reasons:

- To manage Large software
- For more Scalability
- Cost Management
- To manage the dynamic nature of software
- For better quality Management

Need of Software Engineering

The necessity of software engineering appears because of a higher rate of progress in user requirements and

the environment on which the program is working.

- **Huge Programming:** It is simpler to manufacture a wall than to a house or building, similarly, as the measure of programming become extensive engineering has to step to give it a scientific process.
- **Adaptability:** If the software procedure were not based on scientific and engineering ideas, it would be simpler to re-create new software than to scale an existing one.
- **Cost:** As the hardware industry has demonstrated its skills and huge manufacturing has let down the cost of computer and electronic hardware. But the cost of programming remains high if the proper process is not adapted.
- **Dynamic Nature:** The continually growing and adapting nature of programming hugely depends upon the environment in which the client works. If the quality of the software is continually changing, new upgrades need to be done in the existing one.
- **Quality Management:** Better procedure of software development provides a better and quality software product.

Characteristics of a good software engineer

The features that good software engineers should possess are as follows:

- Exposure to systematic methods, i.e., familiarity with software engineering principles.
- Good technical knowledge of the project range (Domain knowledge).
- Good programming abilities.
- Good communication skills. These skills comprise of oral, written, and interpersonal skills.
- High motivation.
- Sound knowledge of fundamentals of computer science.
- Intelligence.
- Ability to work in a team
- Discipline, etc.

Importance of Software Engineering

The importance of Software engineering is as follows:

1. **Reduces complexity.**
2. **To minimize software cost**
3. **To decrease time**
4. **Handling big projects**
5. **Reliable software**

Video Content / Details of website for further learning (if any):

https://www.javatpoint.com/software-engineering-tutorial

https://www.youtube.com/watch?v=IzJzUruMpeE

Important Books/Journals for further learning including the page nos.:

Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-pg:241

21/2/2021

Course Teacher

Verified by HOD

L-2

LECTURE HANDOUTS

CSE | **II/IV**

Course Name with Code : 19CSC12/Software Engineering

Course Teacher : V.Karuppuchamy

Unit : I- Software Process And Agile Development Date of Lecture : 24/2/2021

Topic of Lecture : Software Process

Introduction : (Maximum 5 sentences) :

- A software process is a collection of activities, actions, and tasks that are required to build high-quality software.
- A process defines who is doing what when and how to reach a certain goal.
- The aim of software process is effective on-time delivery of software with quality.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Software Basics
- Physical components
- Software Myths
- Layered Technology

Detailed content of the Lecture:

Elements of a software Process

Activity:

➢ An activity helps to achieve a broad objective (e.g., communication with stakeholders) and is applied regardless of the application domain, size, complexity and degree of rigor with which software engineering is to be applied.

Action

➢ Actions consist of the set of tasks that is used to produce the product.

Task

➢ A task focuses on a small, but well-defined objective that produces a tangible outcome.

Process framework

➢ A process framework establishes the foundation for a complete software engineering process by identifying a small number of framework activities that are applicable to all software projects.

Process framework activities:

Communication

➢ Before starting any technical work, it is important to communicate and collaborate with the customer and other stakeholders.

➢ The main objective is to understand stakeholders' objectives for the project and to gather requirements that helps to define software features and functions.

➢ Software project plan defines the software engineering work by describing the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.

Modeling

➢ Model helps to better understand software requirements and the design that will achieve those requirements.

Construction

➢ This activity combines code generation and the testing that is required to uncover the errors in the code.

Deployment

➢ The software is delivered to the customer, who evaluates the product and provides feedback based on the evaluation.

➢ Generally, the framework activities are applied iteratively as a project progresses.

➢ Each project iteration produces a software increment that provides stakeholders with a subset of overall software features and functionality.

➢ As each increment is produced, the software becomes more and more complete.

➢ The Process framework activities are complemented by a number of umbrella activities which helps to manage and control progress, quality, change, and risk.

Umbrella activities:

- **Software project tracking and control** helps to assess progress against the project plan and take any necessary action to maintain the schedule.
- **Risk management** assesses risks that may affect the outcome and quality of the project.
- **Software quality assurance** defines the activities required to ensure software quality.
- **Technical reviews** assess the products to uncover and remove errors before they are propagated to the next activity.
- **Measurement** defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders' needs.
- **Software configuration management** manages the effects of change throughout the software process.
- **Reusability management** defines criteria for work product reuse and establishes mechanisms to achieve reusable components.
- **Work product preparation and production** encompasses the activities required to create work products such as models, documents, and lists.

Video Content / Details of website for further learning (if any):

https://www.javatpoint.com/software-engineering-tutorial

ht .://www.youtube.com/watch?v=IzJzUruMpeE

Important Books/Journals for further learning including the page nos.:

Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-pg: 247

22/2/2024

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu
Estd. 2000
IQAC

L-3

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : 19CSC12/Software Engineering

Course Teacher : V.Karuppuchamy

Unit : I- Software Process And Agile Development **Date of Lecture :** 25/2/2021

Topic of Lecture : Process Model

Introduction : (Maximum 5 sentences) :

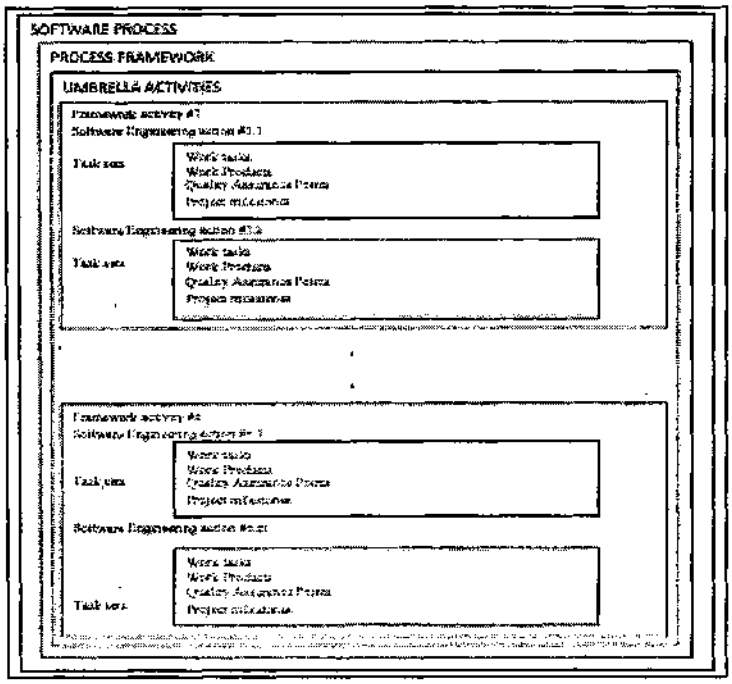
- Process flow describes how the framework activities and the actions and tasks that occur within each framework activity are organized with respect to sequence and time
- Linear process flow executes each of the five activities in sequence.
- An iterative process flow repeats one or more of the activities before proceeding to the next.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Software Basics
- Software Process
- Software Myths
- Layered Technology

Detailed content of the Lecture:



Process Patterns

- A process pattern provides a template for describing problem solutions that is encountered during software engineering work
- The template for describing a process pattern:
  - **Pattern Name.** - a meaningful name describing the pattern within the context (e.g., **Technical Reviews**).
  - **Forces** - The environment where the pattern is encountered.
  - **Type** - The pattern type is specified. They are three types of patterns:

1. **Stage patterns** define a problem associated with a framework activity for the process. A stage pattern has multiple related task patterns. Eg: Establishing Communication includes the task pattern Requirements Gathering and others.
2. **Task patterns** define a problem associated with a software engineering action or work task and relevant to successful software engineering practice.
3. **Phase patterns** define the sequence of framework activities that occur with the process. Eg: Sprial Model or Prototyping

  - **Initial context** - This describes the conditions under which the pattern applies.
  - **Problem** - The specific problem to be solved by the pattern.
  - **Solution** - Describes how to implement the pattern successfully and how the initial state of the process is modified as a consequence of the initiation of the pattern.
  - **Resulting Context** - Describes the conditions that will result once the pattern has been successfully implemented.
  - **Related Patterns - Provide** a list of all process patterns that are directly related to this one. Eg: the stage pattern Communication encompasses the task patterns: Project Team, Collaborative Guidelines, Scope Isolation, Requirements Gathering, Constraint Description, and Scenario Creation.
  - **Known Uses and Examples - Indicate** the specific instances in which the pattern is applicable.

- Process patterns provide an effective mechanism for addressing problems associated with any software process.
- After developed the process patterns can be reused for the definition of process variants.

Video Content / Details of website for further learning (if any):

https://www.javatpoint.com/software-engineering-tutorial

https://www.youtube.com/watch?v=IzJzUruMpeE

Important Books/Journals for further learning including the page nos.:

Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-pg: 251

24/12/2021

Course Teacher

24/12

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-4

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : 19CSC12/Software Engineering

Course Teacher : V.Karuppuchamy

Unit : I- Software Process And Agile Development **Date of Lecture :** 26/2/2021

Topic of Lecture : Perspective Process Models

Introduction : (Maximum 5 sentences) :

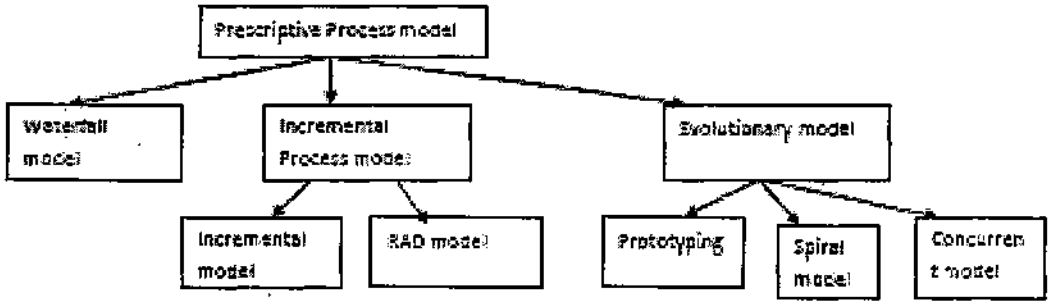
- Prescriptive process models stress detailed definition, identification, and application of process activities and tasks.
- Their intent is to improve system quality, make projects more manageable, make delivery dates and costs more predictable, and guide teams of software engineers as they perform the work required to build a system.

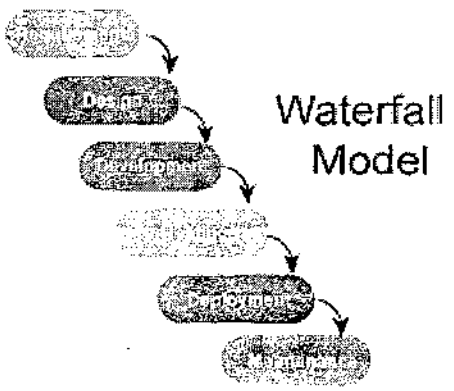
Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Software Basics
- Software Myths
- Basic of Software Engineering
- Process Model

Detailed content of the Lecture:





- This is also referred as classic life cycle or linear - sequential model
- It is the oldest paradigm for SE.
- This model is used for developing projects where the requirements are well defined and reasonably stable, it leads to a linear fashion.
- Waterfall model suggests a systematic, sequential approach to software development.

V-Model

- A variation of waterfall model is referred as V model which includes the quality assurance actions associated with communication, modeling and early code construction activities.
- Team first moves down the left side of the V to refine the problem requirements.
- Once code is generated, the team moves up the right side of the V, performing a series of tests that validate each of the models created.
- The V-model provides a way of visualizing how verification and validation actions are applied at the different stages of development.

Incremental Process Models

- Incremental models construct a partial implementation of a total system and then slowly add increased functionality
- The incremental model prioritizes requirements of the system and then implements them in groups.

R (Rapid Application Development)

- RAD model is based on prototyping and iterative development with no specific planning involved.
- Different phases of RAD model include

Evolutionary Models

- In some cases the requirement changes over time, for such projects evolutionary approach can be used where a limited version is delivered to meet competitive pressure.
- In this model, a set of core product and requirements is well understood, but the details and extension have yet to be defined.
- It is iterative that enables to develop increasingly more complete version of the software.
- Two types of evolutionary approaches are there namely
  - Prototyping and
  - Spiral models.

Concurrent Model

- Allow a software team to represent iterative and concurrent elements of any of the process models. For example, the modeling activity defined for the spiral model is accomplished by invoking one or more of the following actions: prototyping, analysis and design.

Video Content / Details of website for further learning (if any):

https://www.javatpoint.com/software-engineering-tutorial

https://www.youtube.com/watch?v=IzJzUruMpeE

Important Books/Journals for further learning including the page nos.:

Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-pg: 285

(K)
25/2/2021
Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu
Estd. 2000
IQAC

L-5

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : 19CSC12/Software Engineering

Course Teacher : V.Karuppuchamy

Unit : I- Software Process And Agile Development Date of Lecture : 27/2/2021

Topic of Lecture : Specialized Process Models

Introduction : (Maximum 5 sentences) :

- These models tend to be applied when a specialized or narrowly defined software engineering approach is chosen.
- Their intent is to improve system quality, make projects more manageable, make delivery dates and costs more predictable, and guide teams of software engineers as they perform the work required to build a system.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Software Basics
- Software Myths
- Process Model
- Perspective Process Model

Detailed content of the Lecture:

Component-Based Development

- The component-based development model incorporates many of the characteristics of the spiral model.
- It is evolutionary in nature, uses an iterative approach to the creation of software.
- The component-based development model constructs applications from prepackaged software components.
- Modeling and construction activities begin with the identification of candidate components.
- These components can be designed as either conventional software modules or object - oriented classes or packages of classes.

The Formal Methods Model

- Every software engineering methodology is based on a recommended development process proceeding

- Requirements, Specification, Design
- Coding, Unit Testing
- Integration and System Testing, Maintenance

➢ Formal methods can

- Be a foundation for designing safety critical systems
- Be a foundation for describing complex systems
- Provide support for program development

➢ The formal methods model has a set of activities that leads to formal mathematical specification of computer software.

Aspect-Oriented Software Development

➢ Aspect-oriented software development (AOSD) aims to address crosscutting concerns by providing means for systematic identification, separation, representation and composition.

➢ Crosscutting concerns are encapsulated in separate modules, known as aspects, so that localization can be promoted. This results in better support for modularization hence reducing development, maintenance and evolution costs.

➢ Some Crosscutting concerns are high-level properties of a system (e.g., security, fault tolerance). Other concerns affect functions (e.g., the application of business rules), while others are systemic (e.g., task synchronization or memory management).

➢ Aspect-oriented software development (AOSD) is often referred as aspect-oriented programming (AOP), provides a process and methodological approach for defining, specifying, designing, and constructing aspects.

➢ An aspect-oriented process is likely to adopt both evolutionary and concurrent process models.

➢ The evolutionary model is appropriate as aspects are identified and then constructed.

➢ The parallel nature of concurrent development is essential because aspects are engineered independently of localized software components.

Video Content / Details of website for further learning (if any):

https://www.javatpoint.com/software-engineering-tutorial

https://www.youtube.com/watch?v=IzJzUruMpeE

Important Books/Journals for further learning including the page nos.:

Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-pg: 295

26/12/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu
Estd. 2000
IQAC

L-6

LECTURE HANDOUTS

CSE | **II/IV**

Course Name with Code : 19CSC12/Software Engineering

Course Teacher : V.Karuppuchamy

Unit : I- Software Process And Agile Development Date of Lecture : 11.08.21

Topic of Lecture : Introduction to Agility

Introduction : (Maximum 5 sentences) :

- Agility is effective (rapid and adaptive) response to change (changes in team members, new technology, requirements etc) that may have an impact on the product being developed.
- It encourages effective communication in structure and attitudes among all team members, technological and business people, software engineers and managers.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Software Basics
- Process Model
- Perspective Process Model
- Specialized Process Model

Detailed content of the Lecture:

Agility and the Cost of Change

➢ The cost of change **increases nonlinearly** as a project progress.

➢ It is easy to accommodate a change in the early stages of a project and cost will be minimal but in case of major functional change, it requires a modification to the architectural design, construction, testing etc. so the Costs escalate quickly.

Agility Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

face–to–face conversation.

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity – the art of maximizing the amount of work not done – is essential.

11. The best architectures, requirements, and designs emerge from self–organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Human Factors

- **"Agile development focuses on the talents and skills of individuals, molding the process to specific people and teams." i.e. The process molds to the needs of the people and team, not the other way around.**
- The key traits that must exist among the people on an agile team are:
  - Competence.
  - Common focus.
  - Collaboration.
  - Decision-making ability.
  - Fuzzy problem-solving ability.
  - Mutual trust and respect.
  - Self-organization.

Video Content / Details of website for further learning (if any):

https://www.javatpoint.com/software-engineering-tutorial

https://www.youtube.com/watch?v=IzJzUruMpeE

Important Books/Journals for further learning including the page nos.:

Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-pg: 305

27/2/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu
Estd. 2000
IQAC

L-7

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : 19CSC12/Software Engineering

Course Teacher : V.Karuppuchamy

Unit : I- Software Process And Agile Development Date of Lecture : 1/3/21

Topic of Lecture : Agile Process

Introduction : (Maximum 5 sentences) :

- It recognizes that planning in an uncertain world has its limits and that a project plan must be flexible.
- Agility can be applied to any software process but it is essential that the process should support incremental delivery strategy that gets working software to the customer as rapidly as feasible for the product type and operational environment.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Software Basics
- Perspective Process Model
- Specialized Process Model
- Agile Process

Detailed content of the Lecture:

Agile Process

➢ Agile software process addresses a number of key assumptions say:

1. It is difficult to predict in advance which software requirements will persist and which will change. It is equally difficult to predict how customer priorities will change as the project proceeds.
2. For many types of software, design and construction are interleaved. It is difficult to predict how much design is necessary before construction is used to prove the design.
3. Analysis, design, construction, and testing are not as predictable (from a planning point of view) as we might like.

➢ An agile process must be **adaptable** in order to manage the unpredictability.

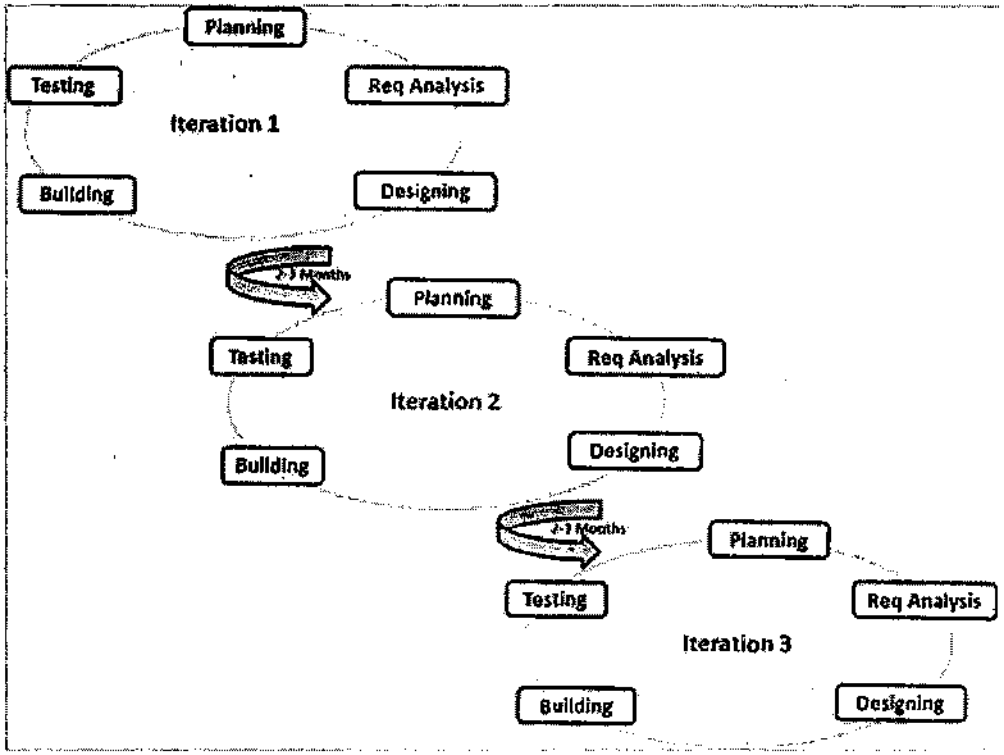
➢ An agile software process must **adapt incrementally**.

➢ Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.

➢ Agile Methods break the product into small incremental builds.

required by the customer.

- Every iteration involves cross functional teams working simultaneously on various areas like –
  - Planning
  - Requirements Analysis
  - Design
  - Coding
  - Testing
- In Agile methodology, there is no detailed planning and there is clarity only in respect of what features need to be developed. The team adapts to the changing product requirements dynamically.
- The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future.



Video Content / Details of website for further learning (if any):

https://www.javatpoint.com/software-engineering-tutorial

https://www.youtube.com/watch?v=IzJzUruMpeE

Important Books/Journals for further learning including the page nos.:

Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-pg: 312

11/3/2021
Course Teacher

Verified by HOD

L-8

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : 19CSC12/Software Engineering

Course Teacher : V.Karuppuchamy

Unit : I- Software Process And Agile Development Date of Lecture : 1|3|2021

Topic of Lecture : Extreme programming

Introduction : (Maximum 5 sentences) :

- XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop a software.
- Extreme Programming (XP), the most widely used approach to agile software development proposed by Kent Beck.
- Recently, a variant of XP, called Industrial XP (IXP) has been proposed which refines XP and targets the agile process specifically for use within large organizations.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Software Basics
- Perspective Process Model
- Specialized Process Model
- Agile Methodology

Detailed content of the Lecture:

XP Values

➢ XP is based on the 5 values:

1. Communication
2. Simplicity
3. Feedback
4. Courage
5. Respect.

➢ Each of these values is used as a driver for specific XP activities, actions, and tasks.

Communication

➢ To achieve effective communication, XP emphasizes close, informal (verbal) collaboration between customers and developers.

➢ To achieve simplicity, XP restricts developers to design only for immediate needs, rather than consider future needs. The design can be refactored later if necessary.

Feedback

➢ Feedback is derived from three sources: the implemented software itself, the customer, and other software team members.

Courage

➢ Team should be prepared to make hard decisions that support the other principles and practices.

Respect

By following each of these values, the agile team inculcates respect among its members, between other stakeholders and team members, and indirectly, for the software itself.

Industrial XP

IXP incorporates **six new practices** that are designed to help ensure that an XP project works successfully for significant projects within a large organization.

1. Readiness assessment:

The assessment ascertains whether (1) an appropriate development environment exists to support IXP, (2) the team will be populated by the proper set of stakeholders, (3) the organization has a distinct quality program and supports continuous improvement, (4) the organizational culture will support the new values of an agile team, and (5) the broader project community will be populated appropriately.

2. **Project community**:

Classic XP suggests that the right people be used to populate the agile team to ensure success. The people on the team must be well-trained, adaptable and skilled, and have the proper temperament to contribute to a self-organizing team.

3. Project chartering:

Chartering also examines the context of the project to determine how it complements, extends, or replaces existing systems or processes.

Video Content / Details of website for further learning (if any):

https://www.javatpoint.com/software-engineering-tutorial

https://www.youtube.com/watch?v=IzJzUruMpeE

Important Books/Journals for further learning including the page nos.:

Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-pg: 320

13/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-9

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : 19CSC12/Software Engineering

Course Teacher : V.Karuppuchamy

Unit : I- Software Process And Agile Development **Date of Lecture :** 3/3/2021

Topic of Lecture : XP Process

Introduction : (Maximum 5 sentences) :

XP Process

- Extreme Programming uses an object-oriented approach that encompasses a set of rules and practices that occur within the context of four framework activities:
- Planning
- Design
- Coding
- Testing

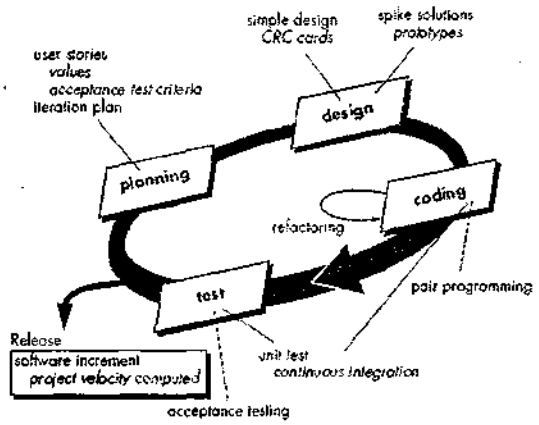
Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Software Basics
- Perspective Process Model
- Specialized Process Model
- Extreme Programming

Detailed content of the Lecture:

XP Process



- The planning activity is also called as planning game.
- Planning begins with a requirements gathering where customers and developers negotiate requirements in the form of User stories captured on index cards.
- User stories describe the required output, features, and functionality for software to be built. Each story is written by the customer and is placed on an index card.
- The customer assigns a value (i.e., a priority) to the story based on the overall business value of the function.
- Agile team assesses each story and assigns a cost measured in development weeks.
- If the story is estimated to require more than three development weeks, the customer is asked to split the story into smaller stories and the assignment of value and cost occurs again.
- Stories are grouped to for a deliverable increment and a commitment is made on delivery date.

XP Design

- XP design follows the KIS (keep it simple) principle.
- A simple design is preferred than complex representation.
- Design provides implementation guidance for the story.
- XP encourages the use of CRC cards. CRC (class-responsibility collaborator) cards identify and organize the object-oriented classes that are relevant to the current software increment.
- Eg: sample CRC for ATM system

XP Coding

- XP recommends the construction of a unit test for a store *before* coding commences.
- A key concept during the coding activity is **"pair programming"**
- This provides a mechanism for real-time problem solving and real-time quality assurance.
- As pair programmers complete their work, the code they develop is integrated with the work of others.
- This "continuous integration" strategy helps to avoid compatibility and interfacing problems and provides a "smoke testing" environment that helps to uncover errors early.

XP Testing

- XP uses Unit and Acceptance Testing.
- All unit tests are executed daily
  - "Acceptance tests" are defined by the customer and executed to assess customer visible functionality
- The unit tests that are created should be implemented using a framework that enables them to be automated.
- This encourages a regression testing strategy whenever code is modified.
- The individual unit tests are organized into a "universal testing suite", integration and validation testing of the system can occur on a daily basis.
- This provides the XP team with a continual indication of progress and also can raise warning flags early if things go awry.

Video Content / Details of website for further learning (if any):

https://www.javatpoint.com/software-engineering-tutorial

https://www.youtube.com/watch?v=IzJzUruMpeE

Important Books/Journals for further learning including the page nos.:

Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-pg: 354

1/3/2021
Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-10

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering/19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **II-Requirement Analysis and Specification** **Date of Lecture :** 4/3/2021

Topic of Lecture : Software Requirements

Introduction : (Maximum 5 sentences) :

- The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product.
- The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- **Software Basics**
- **Physical components**
- **Types of Software**
- **Hardware Basics**

Detailed content of the Lecture:

Requirement Engineering

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

The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

Requirement Engineering Process

It is a four step process, which includes –

- Feasibility Study
- Requirement Gathering
- Software Requirement Specification
- Software Requirement Validation

Let us see the process briefly -

Feasibility study

When the client approaches the organization for getting the desired product developed, it comes up with rough idea about what all functions the software must perform and which all features are expected from the software.

Referencing to this information, the analysts does a detailed study about whether the desired system and its functionality are feasible to develop.

Requirement Gathering

If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

ftware Requirement Specification

SRS is a document created by system analyst after the requirements are collected from various stakeholders.

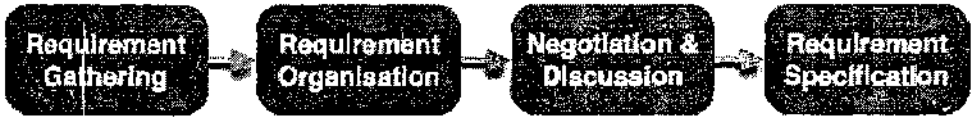
SRS should come up with following features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

Software Requirement Validation

..equirement Elicitation Process

Requirement elicitation process can be depicted using the folloiwng diagram:



- **Requirements gathering** - The developers discuss with the client and end users and know their expectations from the software.
- **Organizing Requirements** - The developers prioritize and arrange the requirements in order of importance, urgency and convenience.
- **Negotiation & discussion** - If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders.

Requirements may then be prioritized and reasonably compromised.

The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.

- **Documentation** - All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

Requirement Elicitation Techniques

Requirements Elicitation is the process to find out the requirements for an intended software system by communicating with client, end users, system users and others who have a stake in the software system development.

There are various ways to discover requirements

Interviews

Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:

- Structured (closed) interviews, where every single information to gather is decided in advance, they follow pattern and matter of discussion firmly.
- Non-structured (open) interviews, where information to gather is not decided in advance, more flexible and less biased.
- Oral interviews
- Written interviews

Surveys

Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

Questionnaires

A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.

A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

Task analysis

Team of engineers and developers may analyze the operation for which the new system is required. If the client already has some software to perform certain operation, it is studied and requirements of proposed

system are collected.

Domain Analysis

Every software falls into some domain category. The expert people in the domain can be a great help to analyze general and specific requirements.

Brainstorming

An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

Prototyping

Prototyping is building user interface without adding detail functionality for user to interpret the features of intended software product. It helps giving better idea of requirements. If there is no software installed at client's end for developer's reference and the client is not aware of its own requirements, the developer creates a prototype based on initially mentioned requirements. The prototype is shown to the client and the feedback is noted. The client feedback serves as an input for requirement gathering.

Observation

Team of experts visit the client's organization or workplace. They observe the actual working of the existing installed systems. They observe the workflow at client's end and how execution problems are dealt. The team itself draws some conclusions which aid to form requirements expected from the software.

Software Requirements Characteristics

Gathering software requirements is the foundation of the entire software development project. Hence they must be clear, correct and well-defined.

A complete Software Requirement Specifications must be:

- Clear
- Correct
- Consistent
- Coherent
- Comprehensible
- Modifiable
- Verifiable
- Prioritized
- Unambiguous
- Traceable

- Credible source

Video Content / Details of website for further learning (if any):

https://www.tutorialspoint.com/software_engineering/software_requirements.htm#:~:text=The%20software%20requirements%20are%20description,from%20client's%20point%20of%20view.

https://www.youtube.com/watch?v=mGkkZoFc-4I

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010,Pg no:154

3/3/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-11

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering/19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **II-Requirement Analysis and Specification** **Date of Lecture :** 5/3/2021

Topic of Lecture : Functional and Non-Functional Requirement

Introduction : (Maximum 5 sentences) :

- A functional requirement defines a system or its component whereas a non-functional requirement defines the performance attribute of a software system.
- Functional requirements along with requirement analysis help identify missing requirements while the advantage of Non-functional requirement is that it helps you to ensure good user experience and ease of operating the software.
- Functional Requirement is a verb while Non-Functional Requirement is an attribute

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements**

Detailed content of the Lecture:

What is a Functional Requirement?

In software engineering, a functional requirement defines a system or its component. It describes the functions a software must perform. A function is nothing but inputs, its behavior, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform.

Functional software requirements help you to capture the intended behavior of the system. This behavior may be expressed as functions, services or tasks or which system is required to perform.

What is Non-Functional Requirement?

A non-functional requirement defines the quality attribute of a software system. They represent a set of

standards used to judge the specific operation of a system. Example, how fast does the website load?

A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system. Failing to meet non-functional requirements can result in systems that fail to satisfy user needs.

Non-functional Requirements allows you to impose constraints or restrictions on the design of the system across the various agile backlogs. Example, the site should load in 3 seconds when the number of simultaneous users is > 10000. Description of non-functional requirements is just as critical as a functional requirement.

Example of Functional Requirements

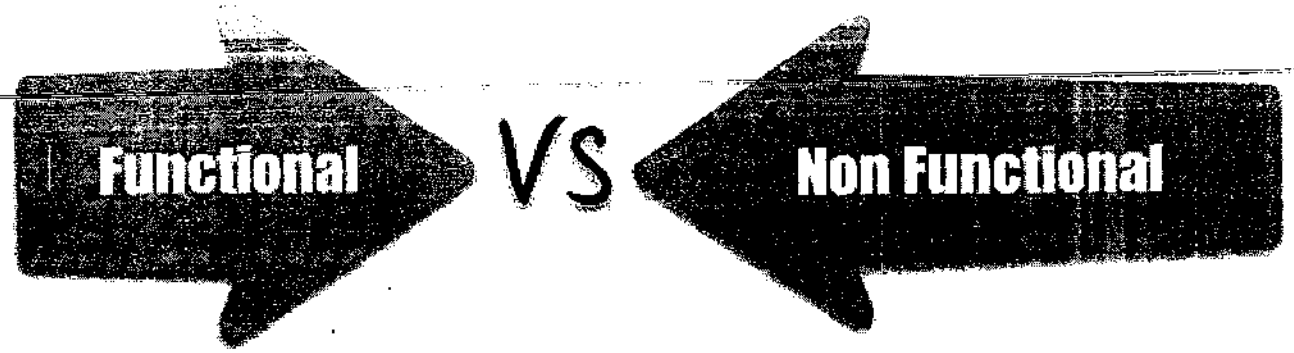
- The software automatically validates customers against the ABC Contact Management System
- The Sales system should allow users to record customers sales
- The background color for all windows in the application will be blue and have a hexadecimal RGB color value of 0x0000FF.
- Only Managerial level employees have the right to view revenue data.
- The software system should be integrated with banking API

Examples of Non-functional requirements

Here, are some examples of non-functional requirement:

1. Users must change the initially assigned login password immediately after the first successful login. Moreover, the initial should never be reused.
2. Employees never allowed updating their salary information. Such attempt should be reported to the security administrator.
3. Every unsuccessful attempt by a user to access an item of data shall be recorded on an audit trail.
4. A website should be capable enough to handle 20 million users with affecting its performance
5. The software should be portable. So moving from one OS to other OS does not create any problem.
6. Privacy of information, the export of restricted technologies, intellectual property rights, etc. should be audited.

Functional vs Non Functional Requirements



Parameters	Functional Requirement	Non-Functional Requirement
What it is	Verb	Attributes
Requirement	It is mandatory	It is non-mandatory
Capturing type	It is captured in use case.	It is captured as a quality attribute.
End-result	Product feature	Product properties
Capturing	Easy to capture	Hard to capture
Objective	Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Area of focus	Focus on user requirement	Concentrates on the user's expectation.
Documentation	Describe what the product does	Describes how the product works
Type of Testing	Functional Testing like System, Integration, End to End, API testing, etc.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc.
Test Execution	Test Execution is done before non-functional testing.	After the functional testing
Product Info	Product Features	Product Properties

Advantages of Functional Requirement

Here, are the pros/advantages of creating a typical functional requirement document-

- Helps you to check whether the application is providing all the functionalities that were mentioned in the functional requirement of that application
- A functional requirement document helps you to define the functionality of a system or one of its subsystems.
- Functional requirements along with requirement analysis help identify missing requirements. They help clearly define the expected system service and behavior.
- Errors caught in the Functional requirement gathering stage are the cheapest to fix.
- Support user goals, tasks, or activities for easy project management
- Functional requirement can be expressed in Use Case form or user story as they exhibit externally

visible functional behavior.

Advantages of Non-Functional Requirement

Benefits/pros of Non-functional testing are:

- The nonfunctional requirements ensure the software system follow legal and compliance rules.
- They ensure the reliability, availability, and performance of the software system
- They ensure good user experience and ease of operating the software.
- They help in formulating security policy of the software system.

Video Content / Details of website for further learning (if any):

https://www.tutorialspoint.com/software_engineering/software_requirements.htm#:~:text=The%20software%20requirements%20are%20description,from%20client's%20point%20of%20view.

https://www.youtube.com/watch?v=j4WITZFLkUM

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:123

4/3/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu
Estd. 2000
IQAC

L-12

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering/19CSC12**
Course Teacher : **V.Karuppuchamy**
Unit : **II-Requirement Analysis and Specification** **Date of Lecture :** 8/3/2021

Topic of Lecture : User requirements, System requirements

Introduction : (Maximum 5 sentences) :

- User requirements, often referred to as user needs, describe what the user does with the system, such as what activities that users must be able to perform.
- User requirements are generally documented in a User Requirements Document (URD) using narrative text.
- User requirements are generally signed off by the user and used as the primary input for creating system requirements.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements**

Detailed content of the Lecture:

User requirements

- User requirements, often referred to as user needs, describe what the user does with the system, such as what activities that users must be able to perform.
- User requirements are generally documented in a User Requirements Document (URD) using narrative text.
- An important and difficult step of designing a software product is determining what the user actually wants it to do. This is because the user is often not able to communicate the entirety of their needs and wants, and the information they provide may also be incomplete, inaccurate and self-conflicting. The responsibility of completely understanding what the customer wants falls on the business analyst. This is why user requirements are generally considered separately from system requirements. The business analyst carefully analyzes user requirements and carefully constructs and documents a set of high quality system requirements ensuring that that the requirements meet certain quality characteristics.

- System requirements are the building blocks developers use to build the system. These are the traditional "shall" statements that describe what the system "shall do." System requirements are classified as either functional or supplemental requirements. A functional requirement specifies something that a user needs to perform their work. For example, a system may be required to enter and print cost estimates; this is a functional requirement. Supplemental or non-functional requirements specify all the remaining requirements not covered by the functional requirements. I prefer to use the term supplemental requirements instead of non-functional requirements; who wants to be termed nonfunctional? Supplemental requirements are sometimes called quality of service requirements. The plan for implementing functional requirements is detailed in the system design. The plan for implementing supplemental requirements is detailed in the system architecture. The list below shows various types of supplemental requirements.
- Accessibility
- Accuracy
- Audit, control, and reporting
- Availability
- Backup and restore
- Capacity, current and forecast
- Certification
- Compliance
- Compatibility of software, tools, standards, platform, database, and the like
- Concurrency
- Configuration management
- Dependency on other parties
- Deployment
- Documentation
- Disaster recovery
- Efficiency (resource consumption for given load)
- Effectiveness (resulting performance in relation to effort)
- Emotional factors (like fun or absorbing)
- Environmental protection
- Error handling
- Escrow
- Exploitability
- Extensibility (adding features, and carry-forward of customizations at next major version upgrade)
- Failure management
- Interoperability
- Legal and regulatory
- Licensing
- Localizability
- Maintainability
- Modifiability
- Network topology
- Open source

- Operability
- Performance/response time
- Price
- Privacy
- Portability
- Quality
- Recovery/recoverability
- Redundancy

System requirements

- System requirements are the configuration that a system must have in order for a hardware or software application to run smoothly and efficiently.
- Failure to meet these requirements can result in installation problems or performance problems. Your audience will listen to you or read the content, but won't do both.
- System requirements often indicate the minimum and the recommended configuration. Hardware system requirements often specify the operating system version, processor type, memory size, available disk space and additional peripherals, if any, needed.
- Software system requirements, in addition to the aforementioned requirements, may also specify additional software dependencies (e.g., libraries, driver version, framework version).

Video Content / Details of website for further learning (if any):

https://enfocussolutions.com/business-user-and-system-requirements/#:~:text=User%20requirements%2C%20often%20referred%20to,(URD)%20using%20narrative%20text.

https://www.youtube.com/watch?v=vpNnZDwC_vs

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-pg no:778

5/8/2021

Course Teacher

Verified by HOD

L-13

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering/19CSC12**
Course Teacher : **V.Karuppuchamy**
Unit : **II-Requirement Analysis and Specification** **Date of Lecture :** 10/3/2021

Topic of Lecture : Software Requirements Document

Introduction : (Maximum 5 sentences) :

- It's the process of writing down the user and system requirements into a document. The requirements should be clear, easy to understand, complete and consistent.
- In practice, this is difficult to achieve as stakeholders interpret the requirements in different ways and there are often inherent conflicts and inconsistencies in the requirements.
- As we've mentioned before, the process in requirements engineering are interleaved, and it's done iteratively. First iteration you specify the user requirements, then, you specify a more detailed system requirements.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements Engineering**

Detailed content of the Lecture:

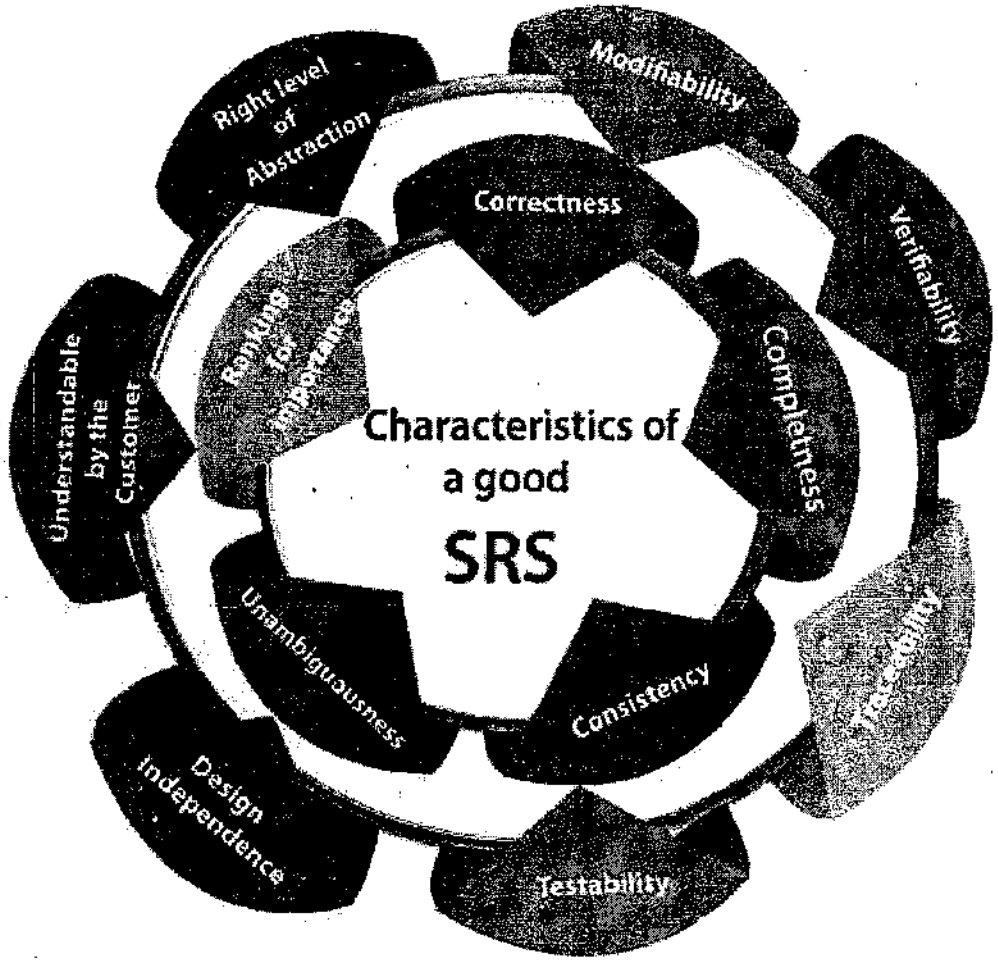
Software Requirement Specifications

The production of the requirements stage of the software development process is **Software Requirements Specifications (SRS)** (also called a **requirements document**). This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited and analyzed. **SRS** is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements.

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

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

Characteristics of good SRS



Following are the features of a good SRS document:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

There are two types of Traceability:

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

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

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

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

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

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

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

Properties of a good SRS document

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

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

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

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

box specification of a system.

Conceptual integrity: It should show conceptual integrity so that the reader can merely understand it.

Response to undesired events: It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.

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

Video Content / Details of website for further learning (if any):

https://www.javatpoint.com/software-requirement-specifications

https://www.youtube.com/watch?v=mGkkZoFc-4I

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010

8/3/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-14

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering/19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **II-Requirement Analysis and Specification** **Date of Lecture :** 11/3/2021

Topic of Lecture : Requirement Engineering Process: Feasibility Studies

Introduction : (Maximum 5 sentences) :

- The process to gather the software requirements from client, analyze and document them is known as requirement engineering.
- The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

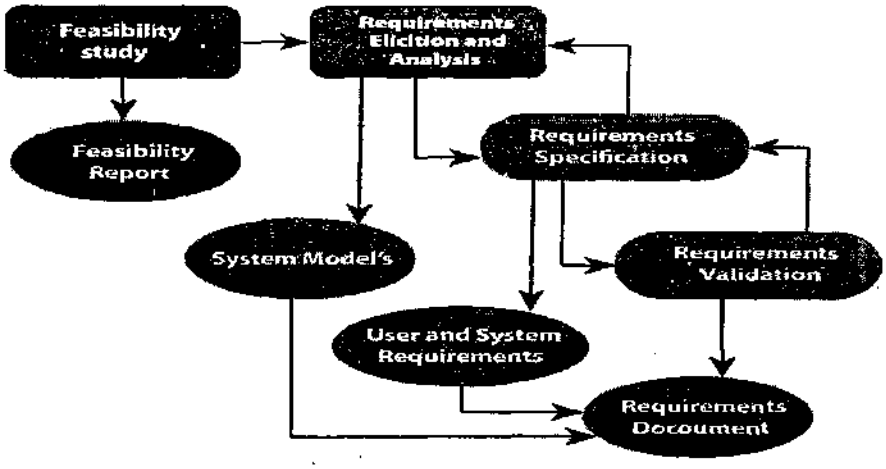
- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements Engineering**

Detailed content of the Lecture:

Requirement Engineering Process

It is a five step process, which includes –

- Feasibility Study
- Requirement Elicitation and Analysis
- Software Requirement Specification
- Software Requirement Validation
- Software Requirement Management



Requirement Engineering Process

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:

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

Video Content / Details of website for further learning (if any):

https://www.javatpoint.com/software-requirement-specifications

https://www.youtube.com/watch?v=mGkkZoFc-4I

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010,Pg no:146

10/3/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-15

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering/19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **II-Requirement Analysis and Specification** **Date of Lecture :** 12/3/2021

Topic of Lecture : Requirements elicitation and analysis, requirements validation, requirements management

Introduction : (Maximum 5 sentences) :

- The process to gather the software requirements from client, analyze and document them is known as requirement engineering.
- The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements Engineering**

Detailed content of the Lecture:

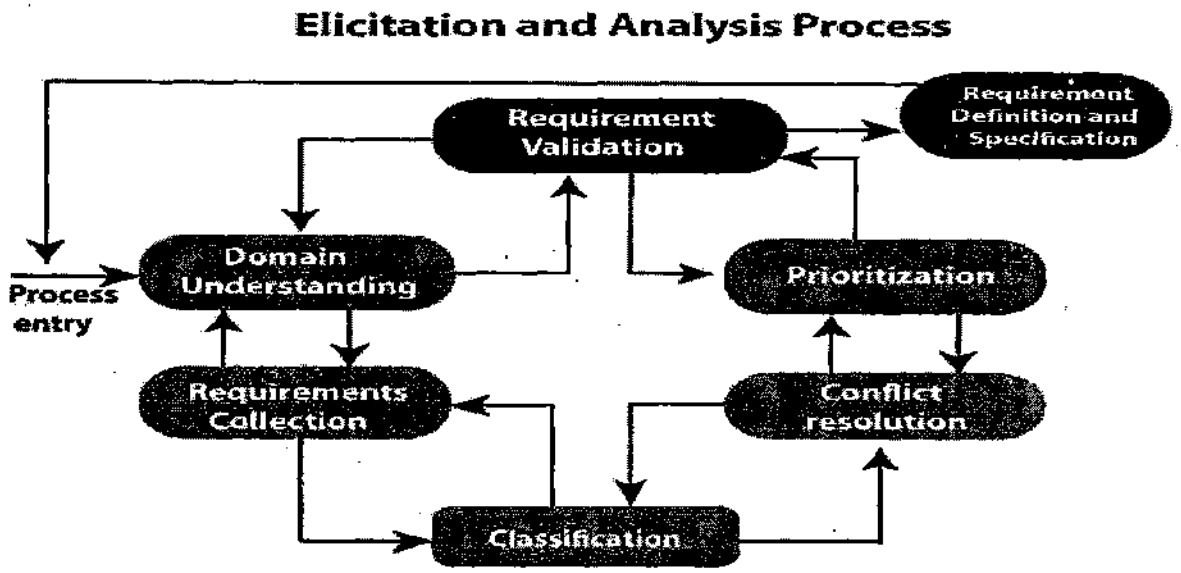
Requirement Elicitation and Analysis:

- ✓ 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.



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.

Requirements validation

After requirement specifications developed, the requirements discussed in this document are validated. The user might demand illegal, impossible solution or experts may misinterpret the needs.

Requirements can be the check against the following conditions -

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

Requirements Validation Techniques

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

Software Requirement Management:

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

Video Content / Details of website for further learning (if any):

https://www.javatpoint.com/software-requirement-specifications

https://www.youtube.com/watch?v=mGkkZoFc-4I

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010,Pg no:824

11/3/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu
Estd. 2000
IQAC

L-16

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering/19CSC12**
Course Teacher : **V.Karuppuchamy**
Unit : **II-Requirement Analysis and Specification** **Date of Lecture :** 13/3/2021

Topic of Lecture : Classical analysis

Introduction : (Maximum 5 sentences) :

- The evaluation of an activity to identify its desired objectives and determine procedures for efficiently attaining them.
- The goal of the **analysis** workflow is to produce a detailed specifications document based on the identified requirements.
- It needs to address both functional and non-functional requirements of the system.
- It is blueprint that designer will use for design ,then programmer to implementation

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements Engineering**

Detailed content of the Lecture:

The goal of the analysis workflow is to produce a detailed specifications document based on the identified requirements.

Specifications doc is significant because:

- this is contract between developer and client regarding what the system will do
- if developer and client are different organizations, it is a legal contract
- it needs to address both functional and non-functional requirements of the system
- it is blueprint that designer will use for design then programmer to implement

The specifications document thus must be *detailed, unambiguous,* and *complete* model of the system, stating what it will do.

Due to ambiguities of specifications written in prose, written descriptions are supplemented or replaced by diagrams.

Historical progression of modeling methods

Models have been used in systems analysis since the 1960s. Historical view of modeling orientations for systems analysis and design

- 1960s: output-oriented
- 1970s: function- and process-oriented
- 1980s: data-oriented
- 1990s: object-oriented
- 2000s: aspect-oriented?

The classical methods are rooted in the 1970s and 1980s.

1970s: Function-oriented

- top-down decomposition of business process
- each decomposition results in set of two or more simpler subprocesses
- recursively decompose until function becomes trivial or easily understood/expressed
- results in tree structure that resembles organizational chart

1970s: Process-oriented

- first apply functional decomposition to identify processes/subprocesses
- use arrows to indicate directed activity
- control flow from process to process is a directed activity
- data flow is a directed activity (e.g. from database to process or vice versa)
- control flow typically expressed using *Flowcharts*
- data flow typically expressed using *Data Flow Diagrams (DFD)*
- these methods are basis for *Structured Analysis*, which dominated business systems development in the 1970s
- functional decomposition and flowcharts dominated for technical/scientific systems development
- Here is a short PowerPoint presentation illustrating some DFDs

1980s: data-oriented

- Identify *data entities* in the system
    - "something that has separate and distinct existence in the world of the users and is of interest to the users in that they need to record data about it" (Brown p. 64)

- Entities are identified by certain *nouns* in a system description
- Identify *entity types* by grouping together similar entities. This is an important form of abstraction.
- Each entity is an *occurrence of* its type

- Then determine what *attributes* those entity types have
  - Attributes are the relevant properties that an entity has
  - All entities of a type have the same attributes
  - Different entities of the same type will have different values for some attributes
  - Attribute values will record an entity's *state*
  - An attribute whose value uniquely identifies an entity may be selected as a *key*
  - If no single attribute has unique value, a combination of attributes may be used.
- Then determine what *associations*, or interactions, those entities have with other entities.
  - one entity can take an action with another, or may play a role for another
  - Associations are identified by certain *verbs* or verb phrases
  - associations are also known as *relationships*
- the *entity-relationship diagram* (ERD) was developed for modeling
- consider some examples: car, bookstore, library, table factory

Video Content / Details of website for further learning (if any):

http://faculty.otterbein.edu/psanderson/comp325/notes/lecture10.html#:~:text=The%20goal%20of%20the%20analysis,it%20is%20a%20legal%20contract

https://www.youtube.com/watch?v=mGkkZoFc-4I

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:148

12/3/2021
Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu
Estd. 2000
IQAC

L-17

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering/19CSC12**
Course Teacher : **V.Karuppuchamy**
Unit : **II-Requirement Analysis and Specification** **Date of Lecture :** 15/3/2021

Topic of Lecture : Structured Systems Analysis

Introduction : (Maximum 5 sentences) :

- A specific technique for systems analysis that covers all activities from initial understanding of the problem through to specification and high-level design of the software system.
- It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements Engineering**

Detailed content of the Lecture:

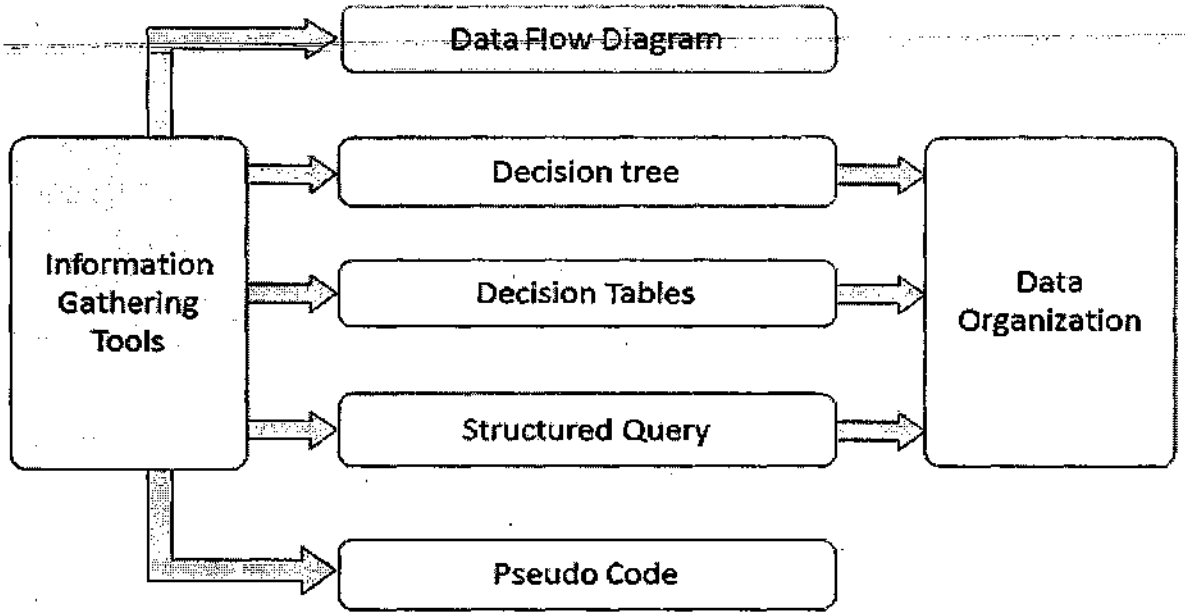
has following attributes –

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

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

- Data Flow Diagrams
- Data Dictionary
- Decision Trees
- Decision Tables
- Structured English

- Pseudocode



Data Flow Diagrams (DFD) or Bubble Chart

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

Symbol Name	Symbol	Meaning
Square		Source or Destination of Data
Arrow		Data flow
Circle		Process transforming data flow
Open Rectangle		Data Store

Context Diagram

- A context diagram helps in understanding the entire system by one DFD which gives the overview of a system.

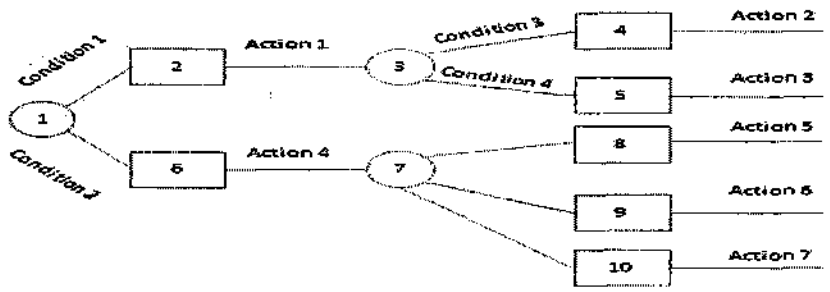
- It starts with mentioning major processes with little details and then goes onto giving more details of the processes with the top-down approach.

Data Dictionary

- A data dictionary is a structured repository of data elements in the system.
- It stores the descriptions of all DFD data elements that is, details and definitions of data flows, data stores, data stored in data stores, and the processes.

Decision Trees

- Decision trees are a method for defining complex relationships by describing decisions and avoiding the problems in communication.
- A decision tree is a diagram that shows alternative actions and conditions within horizontal tree framework. Thus, it depicts which conditions to consider first, second, and so on.



Video Content / Details of website for further learning (if any):

http://faculty.otterbein.edu/psanderson/comp325/notes/lecture10.html#:~:text=The%20goal%20of%20the%20analysis,it%20is%20a%20legal%20contract

https://www.youtube.com/watch?v=mGkkZoFc-4I

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:186

13/3/2021
Course Teacher

13/3
Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu
Estd. 2000
IQAC

L-18

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering/19CSC12**
Course Teacher : **V.Karuppuchamy**
Unit : **II-Requirement Analysis and Specification** **Date of Lecture :** 17/3/2021

Topic of Lecture : Petri Nets- Data Dictionary

Introduction : (Maximum 5 sentences) :

- Petri nets are specific types of modeling constructs useful in data analysis, simulations, business process modeling and other scenarios.
- This type of mathematical construct can help to plan workflows or present data on complicated systems.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

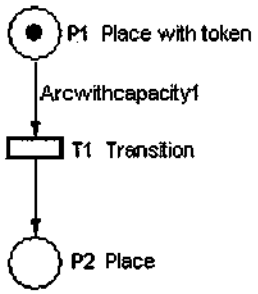
- Software Basics
- Types of Software
- Hardware Basics
- Requirements Engineering

Detailed content of the Lecture:

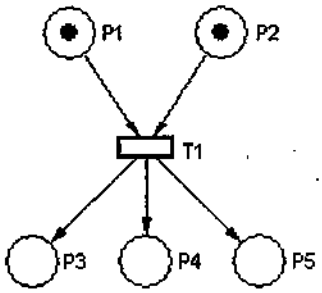
- Petri Nets were developed originally by Carl Adam Petri [Pet62], and were the subject of his dissertation in 1962.
- Since then, Petri Nets and their concepts have been extended and developed, and applied in a variety of areas: Office automation, work-flows, flexible manufacturing, programming languages, protocols and networks, hardware structures, real-time systems, performance evaluation, operations research, embedded systems, defence systems, telecommunications, Internet, e-commerce and trading, railway networks, biological systems.
- This introduction deals with the graphical aspect of Petri Nets for system description, not the algebra of Petri Nets. While the mathematical properties of Petri Nets are interesting and useful, the beginner will find that a good approach is to learn to model systems by constructing them graphically, aided in construction and analysis by computer software for simulation and analysis of Petri Nets.

The Basics:

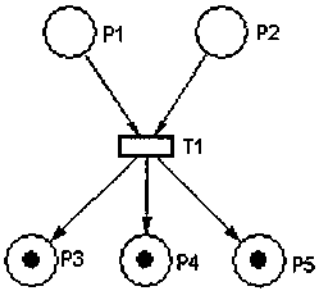
A Petri Net is a collection of directed arcs connecting places and transitions. Places may hold tokens. The state or marking of a net is its assignment of tokens to places. Here is a simple net containing all components of a Petri Net:



- Arcs have capacity 1 by default; if other than 1, the capacity is marked on the arc. Places have infinite capacity by default, and transitions have no capacity, and cannot store tokens at all. With the rule that arcs can only connect places to transitions and vice versa, we have all we need to begin using Petri Nets. A few other features and considerations will be added as we need them.
- A transition is enabled when the number of tokens in each of its input places is at least equal to the arc weight going from the place to the transition. An enabled transition may fire at any time. When fired, the tokens in the input places are moved to output places, according to arc weights and place capacities. This results in a new marking of the net, a state description of all places.

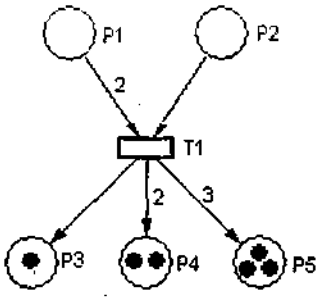


Enabled.ready to file

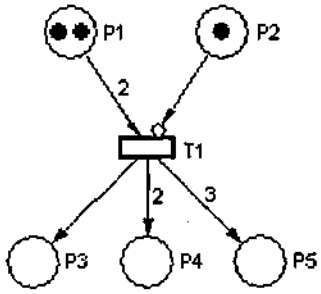


Firing complete

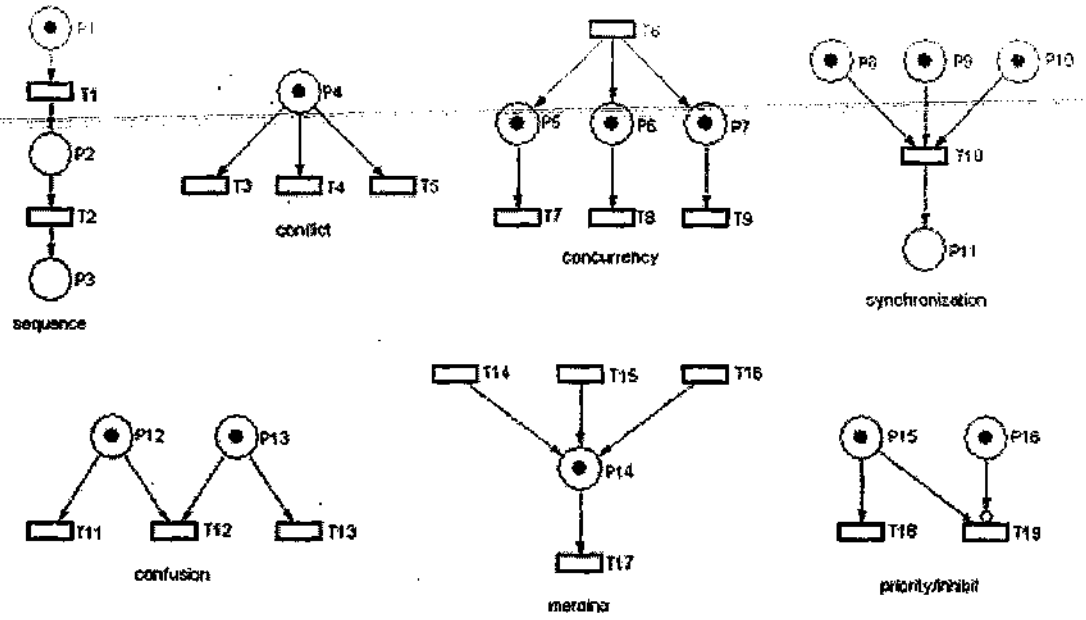
- When arcs have different weights, we have what might at first seem confusing behavior. Here is a similar net, ready to fire: and here it is after firing:



- When a transition fires, it takes the tokens that enabled it from the input places; it then distributes tokens to output places according to arc weights. If the arc weights are all the same, it appears that tokens are moved across the transition. If they differ, however, it appears that tokens may disappear or be created. That, in fact, is what happens; think of the transition as removing its enabling tokens and producing output tokens according to arc weight.
- A special kind of arc, the inhibitor arc, is used to reverse the logic of an input place. With an inhibitor arc, the absence of a token in the input place enables, not the presence:



- This transition cannot fire, because the token in P2 inhibits it.
- Here is a collection of primitive structures that occur in real systems, and thus we find in Petri Nets.



- Sequence is obvious - several things happen in order. Conflict is not so obvious. The token in P4 enables three transitions; but when one of them fires, the token is removed, leaving the remaining two disabled. Unless we can control the timing of firing, we don't know how this net is resolved. Concurrency, again, is obvious; many systems operate with concurrent activities, and this models it well. Synchronization is also modeled well using Petri Nets; when the processes leading into P8, P9 and P10 are finished, all three are synchronized by starting P11.
- Confusion is another not so obvious construct. It is a combination of conflict and concurrency. P12 enables both T11 and T12, but if T11 fires, T12 is no longer enabled.

Video Content / Details of website for further learning (if any):

https://www.techfak.uni-bielefeld.de/~mchen/BioPNML/Intro/pnfaq.html

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:194

15/3/2021

Course Teacher

Verified by HOD

L-19

LECTURE HANDOUTS

CSE | **II/IV**

Course Name with Code : **Software Engineering /19CSC12**
Course Teacher : **V.Karuppuchamy**
Unit : **III-Software Design** **Date of Lecture :** 18/3/2021

Topic of Lecture : Design process

Introduction : (Maximum 5 sentences) :

- Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.
- During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document. Hence the aim of this phase is to transform the SRS document into the design document.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements Engineering**

etailed content of the Lecture:

The following items are designed and documented during the design phase:

- Different modules required.
- Control relationships among modules.
- Interface among different modules.
- Data structure among the different modules.
- Algorithms required implementing among the individual modules.

Objectives of Software Design...

- **Correctness:**
  A good design should be correct i.e. it should correctly implement all the functionalities of the system.

- **Efficiency:**

  A good software design should address the resources, time and cost optimization issues.
- **Understandability:**

  A good design should be easily understandable, for which it should be modular and all the modules are arranged in layers.
- **Completeness:**

  The design should have all the components like data structures, modules, and external interfaces, etc.
- **Maintainability:**

  A good software design should be easily amenable to change whenever a change request is made from the customer side..

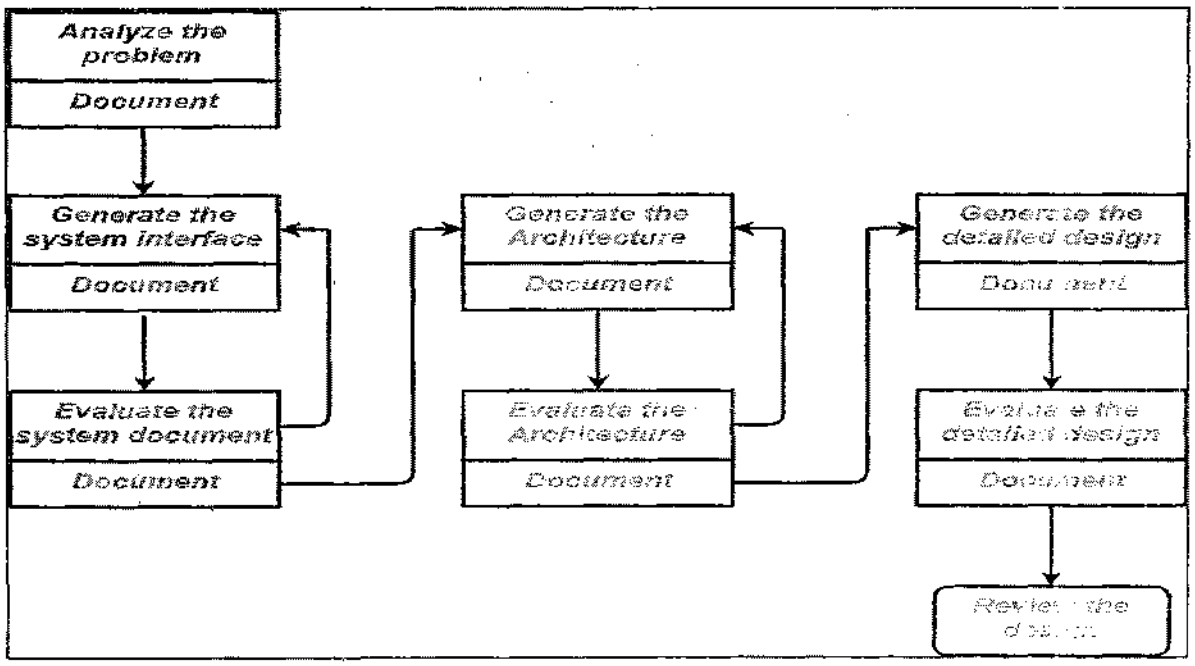
Software Design Process

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language.

The software design process can be divided into the following three

levels of phases of design:

- ✓ Interface Design
- ✓ Architectural Design
- ✓ Detailed Design



- ➢ ***Interface design*** is the specification of the interaction between a system and its environment.
- ➢ This phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e.,

during interface design, the internal of the systems are completely ignored and the system is treated as a black box.

- Attention is focused on the dialogue between the target system and the users, devices, and other systems with which it interacts.
- The design problem statement produced during the problem analysis step should identify the people, other systems, and devices which are collectively called *agents*.

- ***Architectural design*** is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them.
- In architectural design, the overall structure of the system is chosen, but the internal details of major components are ignored.

- *Design* is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures.

The detailed design may include:

- Decomposition of major system components into program units.
- Allocation of functional responsibilities to units.
- User interfaces
- Unit states and state changes
- Data and control interaction between units
- Data packaging and implementation, including issues of scope and visibility of program elements
- Algorithms and data structures

Video Content / Details of website for further learning (if any):

http://www.brainkart.com/article/Petri-Nets-Data-Dictionary_9074/

https://www.sciencebuddies.org/science-fair-projects/engineering-design-process/engineering-design-process-steps

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:219

17/3/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-20

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **III-Software Design** **Date of Lecture :** 19/3/2021

Topic of Lecture : Design Concepts

Introduction : (Maximum 5 sentences) :

- Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.
- During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document. Hence the aim of this phase is to transform the SRS document into the design document.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

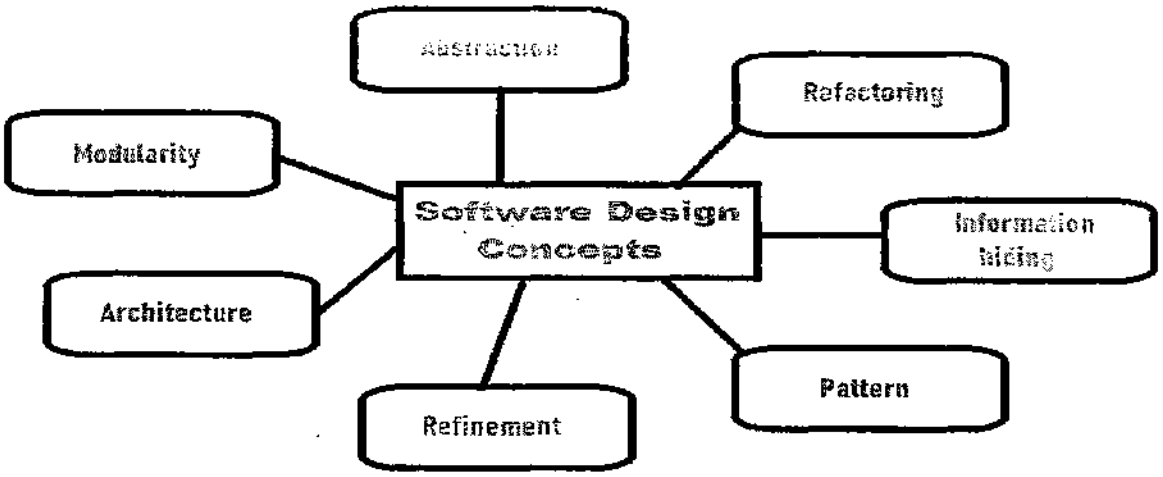
- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements Engineering**

Detailed content of the Lecture:

Software Design Concepts:

- Concepts are defined as a principal idea or invention that comes in our mind or in thought to understand something.
- The software design concept simply means the idea or principle behind the design.
- It describes how you plan to solve the problem of designing software, the logic, or thinking behind how you will design software.
- It allows the software engineer to create the model of the system or software or product that is to be developed or built.
- The software design concept provides a supporting and essential structure or model for developing the right software.

There are many concepts of software design and some of them are given below:



Following points should be considered while designing a Software:

1. Abstraction- hide relevant data

   Abstraction simply means to hide the details to reduce complexity and increases efficiency or quality. Different levels of Abstraction are necessary and must be applied at each stage of the design process so that any error that is present can be removed to increase the efficiency of the software solution and to refine the software solution.

2. **Modularity - subdivide the system**

   Modularity simply means to divide the system or project into smaller parts to reduce the complexity of the system or project. In the same way, modularity in design means to subdivide a system into smaller parts so that these parts can be created independently and then use these parts in different systems to perform different functions. It is necessary to divide the software into components known as modules because nowadays there are different software available like Monolithic software that is hard to grasp for software engineers. So, modularity is design has now become a trend and is also important.

3. **Architecture- design a structure of something**

   Architecture simply means a technique to design a structure of something. Architecture in designing software is a concept that focuses on various elements and the data of the structure. These components interact with each other and use the data of the structure in architecture.

4. **Refinement- removes impurities**

   Refinement simply means to refine something to remove any impurities if present and increase the quality. The refinement concept of software design is actually a process of developing or presenting the software or system in a detailed manner that means to elaborate a system or software. Refinement is very necessary to find out any error if present and then to reduce it.

5. **5.Pattern- a repeated form**

   The pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern. The pattern in the design process means the repetition of a solution to a common recurring problem within a certain context.

6. **Information Hiding- hide the information**

   Information hiding simply means to hide the information so that it cannot be accessed by an unwanted

party. In software design, information hiding is achieved by designing the modules in a manner that the information gathered or contained in one module is hidden and it cant be accessed by any other modules.

7. **Refactoring- reconstruct something**

   Refactoring simply means to reconstruct something in such a way that it does not affect the behavior or any other features. Refactoring in software design means to reconstruct the design to reduce and complexity and simplify it without affecting the behavior or its functions. Fowler has defined refactoring as "the process of changing a software system in a way that it won't affect the behavior of the design and improves the internal structure".

Video Content / Details of website for further learning (if any):

http://www.brainkart.com/article/Petri-Nets-Data-Dictionary_9074/

https://www.delve.com/design-concepts

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:219

18/3/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-21

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **III-Software Design** **Date of Lecture :** 20/3/2021

Topic of Lecture : Design Model

Introduction : (Maximum 5 sentences) :

- A **design model** in Software Engineering is an object-based picture or pictures that represent the use cases for a system.
- Or to put it another way, it is the means to describe a system's implementation and source code in a diagrammatic fashion.
- This type of representation has a couple of advantages.
- First, it is a simpler representation than words alone.
- Second, a group of people can look at these simple diagrams and quickly get the general idea behind a system.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements Engineering**

Detailed content of the Lecture:

Moving from Requirements to Design Specifications

- Moving from requirements (what is needed) to design specifications (a description of what should be built) involves a number of steps.
- These can be different depending on who you talk to, but we'll use the following:
- Architectural - This involves breaking the system down into its major functional pieces and describing each in a diagrammatic fashion. The interaction between each piece is also described. For example, an instant messaging system might have a 'send message' piece and an interaction might be the user types characters that act as input to the send message piece.

- Interface - The architectural interactions are broken down and described in greater detail. For example,

the user typed characters mentioned above might be described as arriving a character at a time, each character is displayed on the cell phone screen as it is typed.

- Component Level - The pieces described in the architectural item above are broken down into lower level components. For example, the send message piece might become 'receive character' component, 'display character' component, and 'transmit character' component.
- Deployment Level - The components arrived at during the previous step are grouped for the purpose of delivery to their final destination. For example, the components of the 'send message' piece might be grouped together with other pieces for deployment on a cell phone.

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.

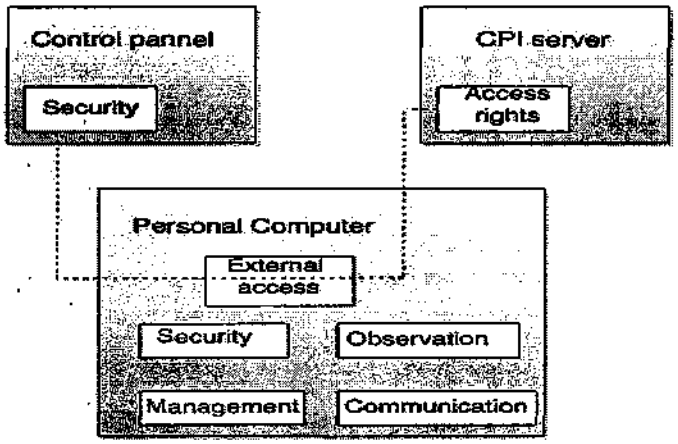


Fig. - Deployment level diagram

Video Content / Details of website for further learning (if any):

http://www.brainkart.com/article/Petri-Nets-Data-Dictionary_9074/

https://study.com/academy/lesson/design-model-in-software-engineering-elements-examples.html

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:233

19/3/2021
Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-22

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **III-Software Design** **Date of Lecture :** 22/3/2021

Topic of Lecture : Design Heuristic

Introduction : (Maximum 5 sentences) :

- Heuristics refers to a non-optimal solution for experience-based techniques to solve problems, learning, and discovery.
- The main goal of heuristic evaluations is to identify any problems associated with the design of user interfaces.
- The simplicity of heuristic evaluation is beneficial at the early stages of design.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements Engineering**

Detailed content of the Lecture:

What is heuristics?

Heuristics refers to a non-optimal solution for experience-based techniques to solve problems, learning, and discovery.

Heuristics of Software Testability

- **Controllability** - Software and hardware states can be controlled by test engineers and the Software modules can be tested independently
- **Observability** - Check for the object or System states and all other factors affecting the output.
- **Availability** - Check if Source code is accessible as product evolves in stages.
- **Simplicity** - Check if the design is consistent. Check for functional simplicity, structural simplicity

and code simplicity.

- **Stability** - Check if the Changes to the software are infrequent and changes are controlled and communicated.

Heuristics Interface for User Interface Design (UID)

- Visibility of system status
- Match between system and the real world
- Consistency and standards
- Error prevention
- Flexibility and efficiency of use
- Aesthetic and minimalist design
- Help and documentation

The main goal of heuristic evaluations is to identify any problems associated with the design of user interfaces. Usability consultant Jakob Nielsen developed this method on the basis of several years of experience in teaching and consulting about usability engineering.

Heuristic evaluations are one of the most informal methods of usability inspection in the field of human-computer interaction. There are many sets of usability design heuristics; they are not mutually exclusive and cover many of the same aspects of user interface design.

Quite often, usability problems that are discovered are categorized—often on a numeric scale—according to their estimated impact on user performance or acceptance. Often the heuristic evaluation is conducted in the context of use cases (typical user tasks), to provide feedback to the developers on the extent to which the interface is likely to be compatible with the intended users' needs and preferences

The simplicity of heuristic evaluation is beneficial at the early stages of design. This usability inspection method does not require user testing which can be burdensome due to the need for users, a place to test them and a payment for their time. Heuristic evaluation requires only one expert, reducing the complexity and expended time for evaluation. Most heuristic evaluations can be accomplished in a matter of days. The time required varies with the size of the artifact, its complexity, the purpose of the review, the nature of the usability issues that arise in the review, and the competence of the reviewers. Using heuristic evaluation prior to user testing will reduce the number and severity of design errors discovered by users. Although heuristic evaluation can uncover many major usability issues in a short period of time, a criticism that is often leveled is that results are highly influenced by the knowledge of the expert reviewer(s). This "one-sided" review repeatedly has different results than software performance testing, each type of testing uncovering a different set of problems.

Video Content / Details of website for further learning (if any):
http://www.brainkart.com/article/Design-Heuristic_9077/
https://www.nngroup.com/videos/heuristic-evaluation/
Important Books/Journals for further learning including the page nos.:
1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:271

20/3/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-23

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **III-Software Design** **Date of Lecture :** 24/3/2021

Topic of Lecture : Architectural styles

Introduction : (Maximum 5 sentences) :

- An architectural style is characterized by the features that make a building or other structure notable or historically identifiable.
- It is a sub-class of style in the visual arts generally, and most styles in architecture related closely to the wider contemporary artistic style.
- A style may include such elements as form, method of construction, building materials, and regional character.
- Most architecture can be classified within a chronology of styles which changes over time reflecting changing fashions, beliefs and religions, or the emergence of new ideas, technology, or materials which make new styles possible.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements Engineering**

Detailed content of the Lecture:

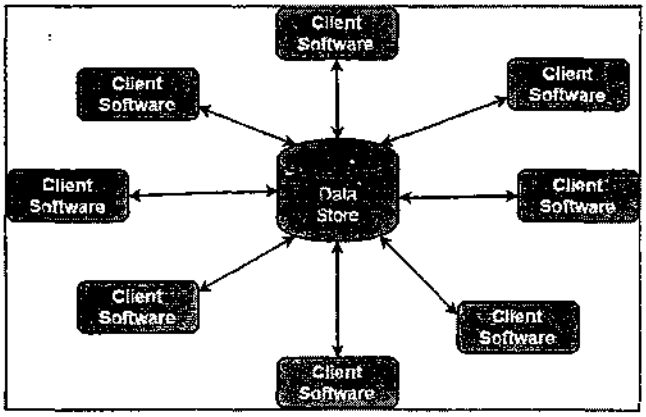
- **The software that is built for computer-based systems can exhibit one of these many architectural**
- **styles.**
- Each style will describe a system category that consists of :
- A set of components(eg: a database, computational modules) that will perform a function required by the system.
- The set of connectors will help in coordination, communication, and cooperation between the components.
- Conditions that how components can be integrated to form the system.

- Semantic models that help the designer to understand the overall properties of the system.
- The use of architectural styles is to establish a structure for all the components of the system.

Taxonomy of Architectural styles :

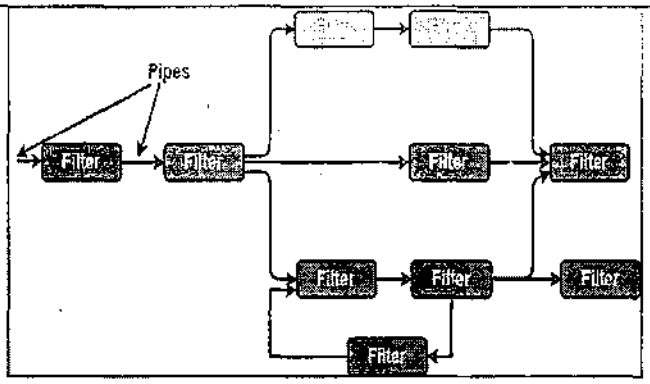
Data centered architectures :

- A data store will reside at the center of this architecture and is accessed frequently by the other components that update, add, delete or modify the data present within the store.
- The figure illustrates a typical data centered style. The client software access a central repository. Variation of this approach are used to transform the repository into a blackboard when data related to client or data of interest for the client change the notifications to client software.
    - Data can be passed among cli ents using blackboard mechanism.



Data flow architectures :

- This kind of architecture is used when input data to be transformed into output data through a series of computational manipulative components.
- The figure represents pipe-and-filter architecture since it uses both pipe and filter and it has a set of components called filters connected by pipes.
- Pipes are used to transmit data from one component to the next.
- Each filter will work independently and is designed to take data input of a certain form and produces data output to the next filter of a specified form. The filters don't require any knowledge of the working of neighboring filters.
- If the data flow degenerates into a single line of transforms, then it is termed as batch sequential. This structure accepts the batch of data and then applies a series of sequential components to transform it.



Call and Return architectures :

- It is used to create a program that is easy to scale and modify. Many sub-styles exist within this category. Two of them are explained below.
- **Remote procedure call architecture:** This components is used to present in a main program or sub program architecture distributed among multiple computers on a network.
- **Main program or Subprogram architectures:** The main program structure decomposes into number of subprograms or function into a control hierarchy. Main program contains number of subprograms that can invoke other components.

Layered architecture:

- A number of different layers are defined with each layer performing a well-defined set of operations. Each layer will do some operations that becomes closer to machine instruction set progressively.
- At the outer layer, components will receive the user interface operations and at the inner layers, components will perform the operating system interfacing(communication and coordination with OS)
- Intermediate layers to utility services and application software functions.

Video Content / Details of website for further learning (if any):

https://en.wikipedia.org/wiki/Architectural_style

https://www.youtube.com/watch?v=TzYYG06x9e0

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:250

22/3/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-24

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **III-Software Design** **Date of Lecture :** 25/3/2021

Topic of Lecture : Architectural Design

Introduction : (Maximum 5 sentences) :

- **Architectural design** is a process for identifying the sub-systems making up a system and the framework for sub-system control and communication.
- The output of this design process is a description of the software architecture.
- Architectural design is an early stage of the system design process.
- It represents the link between specification and design processes and is often carried out in parallel with some specification activities.
- It involves identifying major system components and their communications.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements Engineering**

Detailed content of the Lecture:

Requirements of the software should be transformed into an architecture that describes the software's top-level structure and identifies its components. This is accomplished through architectural design (also called **system design**), which acts as a preliminary 'blueprint' from which software can be developed. **IEEE** defines architectural design as 'the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.' This framework is established by examining the software requirements document and designing a model for providing implementation details. These details are used to specify the components of the system along with their inputs, outputs, functions, and the interaction between them. An architectural design performs the following functions.

1. It defines an abstraction level at which the designers can specify the functional and performance behaviour of the system.

2. It acts as a guideline for enhancing the system (when ever required) by describing those features of the system that can be modified easily without affecting the system integrity.

3. It evaluates all top-level designs.

4. It develops and documents top-level design for the external and internal interfaces.

5. It develops preliminary versions of user documentation.

6. It defines and documents preliminary test requirements and the schedule for software integration.

7. The sources of architectural design are listed below.

8. Information regarding the application domain for the software to be developed

9. Using data-flow diagrams

10. Availability of architectural patterns and architectural styles.

Architectural design is of crucial importance in software engineering during which the essential requirements like reliability, cost, and performance are dealt with. This task is cumbersome as the software engineering paradigm is shifting from monolithic, stand-alone, built-from-scratch systems to componentized, evolvable, standards-based, and product line-oriented systems. Also, a key challenge for designers is to know precisely how to proceed from requirements to architectural design. To avoid these problems, designers adopt strategies such as reusability, componentization, platform-based, standards-based, and so on.

Though the architectural design is the responsibility of developers, some other people like user representatives, systems engineers, hardware engineers, and operations personnel are also involved. All these stakeholders must also be consulted while reviewing the architectural design in order to minimize the risks and errors.

Architectural Design Representation

Architectural design can be represented using the following models.

- **Structural model:** Illustrates architecture as an ordered collection of program components
- **Dynamic model:** Specifies the behavioral aspect of the software architecture and indicates how the structure or system configuration changes as the function changes due to change in the external environment
- **Process model:** Focuses on the design of the business or technical process, which must be implemented in the system
- **Functional model:** Represents the functional hierarchy of a system
- **Framework model:** Attempts to identify repeatable architectural design patterns encountered in

similar types of application. This leads to an increase in the level of abstraction.

Architectural Design Output

The architectural design process results in an **Architectural Design Document (ADD).** This document consists of a number of graphical representations thatcomprises software models along with associated descriptive text. The softwaremodels include static model, interface model, relationship model, and dynamic processmodel. They show how the system is organized into a process at run-time.

Architectural design document gives the developers a solution to the problem stated in the Software Requirements Specification (SRS). Note that it considers only those requirements in detail that affect the program structure. In addition to ADD, other outputs of the architectural design are listed below.

- Various reports including audit report, progress report, and configuration status accounts report
- Various plans for detailed design phase, which include the following
- Software verification and validation plan
- Software configuration management plan
- Software quality assurance plan
- Software project management plan.

Software architectures can be designed at **two levels of abstraction**:

- **Architecture in the small** is concerned with the architecture of individual programs. At this level, we are concerned with the way that an individual program is decomposed into components.
- **Architecture in the large** is concerned with the architecture of complex enterprise systems that include other systems, programs, and program components. These enterprise systems are distributed over different computers, which may be owned and managed by different companies.

Video Content / Details of website for further learning (if any):

https://www.architecturaldigest.in/architecture-design/

https://www.youtube.com/watch?v=ly8orBNiNQM

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:255

24/3/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-25

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **III-Software Design** **Date of Lecture :** 26/3/2021

Topic of Lecture : Architectural Mapping using Data Flow

Introduction : (Maximum 5 sentences) :

- Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.
- During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document. Hence the aim of this phase is to transform the SRS document into the design document.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Software Basics
- Types of Software
- Hardware Basics
- Requirements Engineering

Detailed content of the Lecture:

Architectural Mapping Using Data Flow

- A mapping technique, called structured design, is often characterized as a data flow-oriented design method because it provides a convenient transition from a data flow diagram to software architecture.
- The transition from information flow to program structure is accomplished as part of a six step process:
  - (1) The type of information flow is established,
  - (2) Flow boundaries are indicated,
  - (3) The DFD is mapped into the program structure,
  - (4) Control hierarchy is defined,
  - (5) The resultant structure is refined using design measures.
  - (6) The architectural description is refined and elaborated.
- Example of data flow mapping, a step-by-step "transform" mapping for a small part of the SafeHome

security function.

- In order to perform the mapping,
- The type of information flow must be determined. It is called ***transform flow and exhibits a linear quality.***
- Finally, it flows out of the system along an outgoing flow path that transforms the data into external world form.

Transform Mapping

- Transform mapping is a set of design steps that allows a DFD with transform flow characteristics to be mapped into a specific architectural style.
- **To map these data flow diagrams into a software architecture, you would initiate the following design steps:**
    - Step 1. Review the fundamental system model.
    - Step 2. Review and refine data flow diagrams for the software.
    - Step 3. Determine whether the DFD has transform or transaction flow characteristics.
    - Step 4. Isolate the transform center by specifying incoming and outgoing flow boundaries.
    - Step 5. Perform "first-level factoring."
    - Step 6. Perform "second-level factoring."
- Step 7. Refine the first-iteration architecture using design heuristic for improved software quality.

Video Content / Details of website for further learning (if any):

http://softwareengineeringmca.blogspot.com/2017/07/architectural-mapping-using-data-flow-transform-mapping.html

https://www.conceptdraw.com/examples/architectural-mapping-using-dfd-diagram-for-library-management-system

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:265

25/3/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-26

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **III-Software Design** **Date of Lecture :** 29/3/2021

Topic of Lecture : User Interface Design: Interface analysis, Interface Design

Introduction : (Maximum 5 sentences) :

- The goal of this phase is to define the set of interface objects and actions i.e. Control mechanisms that enable the user to perform desired tasks. Indicate how these control mechanisms affect the system.
- Specify the action sequence of tasks and subtasks, also called a user scenario. Indicate the state of the system when the user performs a particular task.
- Always follow the three golden rules stated by Theo Mandel.
- Design issues such as response time, command and action structure, error handling, and help facilities are considered as the design model is refined. This phase serves as the foundation for the implementation phase.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements Engineering**

Detailed content of the Lecture:

User Interface Design

User interface is the front-end application view to which user interacts in order to use the software. The software becomes more popular if its user interface is:

- Attractive
- Simple to use
- Responsive in short time
- Clear to understand
- Consistent on all interface screens

There are two types of User Interface:

1. **Command Line Interface:** Command Line Interface provides a command prompt, where the user types the command and feeds to the system. The user needs to remember the syntax of the command and its use.
2. **Graphical User Interface:** Graphical User Interface provides the simple interactive interface to interact with the system. GUI can be a combination of both hardware and software. Using GUI, user interprets the software.

Golden Rules:

The following are the golden rules stated by Theo Mandel that must be followed during the design of the interface.

Place the user in control:

- Define the interaction modes in such a way that does not force the user into unnecessary or undesired actions: The user should be able to easily enter and exit the mode with little or no effort.
- Provide for flexible interaction: Different people will use different interaction mechanisms, some might use keyboard commands, some might use mouse, some might use touch screen, etc, Hence all interaction mechanisms should be provided.
- Allow user interaction to be interruptable and undoable: When a user is doing a sequence of actions the user must be able to interrupt the sequence to do some other work without losing the work that had been done. The user should also be able to do undo operation.
- Streamline interaction as skill level advances and allow the interaction to be customized: Advanced or highly skilled user should be provided a chance to customize the interface as user wants which allows different interaction mechanisms so that user doesn't feel bored while using the same interaction mechanism.
- Hide technical internals from casual users: The user should not be aware of the internal technical details of the system. He should interact with the interface just to do his work.
- Design for direct interaction with objects that appear on screen: The user should be able to use the objects and manipulate the objects that are present on the screen to perform a necessary task. By this, the user feels easy to control over the screen.

User Interface Analysis and Design

- Interface Design: The goal of this phase is to define the set of interface objects and actions i.e. Control mechanisms that enable the user to perform desired tasks. Indicate how these control mechanisms affect the system. Specify the action sequence of tasks and subtasks, also called a user scenario. Indicate the state of the system when the user performs a particular task. Always follow the three golden rules stated by Theo Mandel. Design issues such as response time, command and action structure, error handling, and help facilities are considered as the design model is refined. This phase serves as the foundation for the implementation phase.

- **Interface construction and implementation:** The implementation activity begins with the creation of prototype (model) that enables usage scenarios to be evaluated. As iterative design process continues a User Interface toolkit that allows the creation of windows, menus, device interaction, error messages, commands, and many other elements of an interactive environment can be used for completing the construction of an interface.
- **Interface Validation:** This phase focuses on testing the interface. The interface should be in such a way that it should be able to perform tasks correctly and it should be able to handle a variety of tasks. It should achieve all the user's requirements. It should be easy to use and easy to learn. Users should accept the interface as a useful one in their work.

Video Content / Details of website for further learning (if any):

https://www.tutorialspoint.com/software_engineering/software_user_interface_design.htm

https://www.youtube.com/watch?v=TWbMAaDoXqg

Important Books/Journals for further learning including the page nos.:

Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:317

26/3/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-27

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering/19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **III-Software Design** **Date of Lecture :** 31/3/2021

Topic of Lecture : Component level Design: Designing Class based components, traditional Components

Introduction : (Maximum 5 sentences) :

- Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.
- During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document. Hence the aim of this phase is to transform the SRS document into the design document.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements Engineering**

Detailed content of the Lecture:

Introduction

Component-level design is elaborative in nature. It transforms information from requirements and architectural models into a design representation that provides sufficient detail to guide the construction (coding and testing) activity.

- The following steps represent a typical task set for component-level design, when it is applied for an object-oriented system.
- Step 1. Identify all design classes that correspond to the problem domain. Using the requirements and architectural model, each analysis class and architectural component is elaborated…
- Step 2. Identify all design classes that correspond to the infrastructure domain.

1. These classes are not described in the requirements model and are often missing from the architecture model, but they must be described at this point.
2. Classes and components in this category include
3. GUI components (often available as reusable components),

4. Operating system components. Object and data management components

- Step 3. Elaborate all design classes that are not acquired (Obtain) as reusable components. Elaboration requires that all interfaces, attributes, and operations necessary to implement the class be described in detail. Design heuristics (e.g., component cohesion and coupling).
- Step 3 (a) . Specify message details when classes or components collaborate. The requirements model makes use of a collaboration diagram to show how analysis classes collaborate with one another.
- Step 3 (b). Identify appropriate interfaces for each component. Within the context of component-level design, a UML interface is "a group of externally visible (i.e., public) operations. The interface contains no internal structure, it has no attributes, no associations . . .
- Step 3 (c). Elaborate attributes and define data types and data structures required to implement them.In general, data structures and types used to define attributes are defined within the context of the programming language that is to be used for implementation. UML defines an attribute's data type using the following syntax: name : type-expression initial-value {property string} where name is the attribute name, type expression is the data type, initial value is the value that the attribute takes when an object is created, and property-string defines a property or characteristic of the attribute.
- Step 3 (d). Describe processing flow within each operation in detail.This may be accomplished using a programming language-based pseudocode or with a UML activity diagram. Each software component is elaborated through a number of iterations that apply the stepwise refinement concept.
- Step 4. Describe persistent data sources (databases and files) and identify the classes required to manage them. Databases and files normally transcend the design description of an individual component. In most cases, these persistent data stores are initially specified as part of architectural design.
- Step 5. Develop and elaborate behavioral representations for a class or component. UML state diagrams were used as part of the requirements model to represent the externally observable behavior of the system.
- Step 6. Elaborate deployment diagrams to provide additional implementation detail. Deployment diagrams are used as part of architectural.
- Step 7. Refactor every component-level design representation and always consider alternatives.

Traditional components

Sequence implements processing steps that are essential in the specification of any algorithm.

Condition provides the facility for selected processing based on some logical occurrence.

Repetition allows for looping.

Graphical Design Notation

- A picture is worth a thousand words," but it's rather important to know which picture and which 1000 words.
- There is no question that graphical tools, such as the UML activity diagram or the flowchart, provide useful pictorial patterns that readily depict procedural detail.
- The activity diagram allows you to represent sequence, condition, and repetition— all elements of structured programming.

- It is a descendent of an earlier pictorial design representation (still used widely) called a flowchart.
- A flowchart, like an activity diagram, is quite simple pictorially. A box is used to indicate a processing step. A diamond represents a logical condition, and arrows show the flow of control

Video Content / Details of website for further learning (if any):

http://www.brainkart.com/article/Designing-Class-based-components,-traditional-Components_9083/

https://www.youtube.com/watch?v=tp-PmCnCLw0

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:282

29/3/2021

Course Teacher

29/3

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu
Estd. 2000
IQAC

L-28

LECTURE HANDOUTS

CSE | **II/IV**

Course Name with Code : **Software Engineering /19CSC12**
Course Teacher : **V.Karuppuchamy**
Unit : **IV-Testing And Implementation** **Date of Lecture :** 1/4/2021

Topic of Lecture : Software testing fundamentals

Introduction : (Maximum 5 sentences) :

- Testing is the process of finding the errors in the software before delivering to the end user.
- The main objective of testing is to uncover errors with a minimum of effort and time.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Functional simplicity
- Structural simplicity
- Code simplicity

Detailed content of the Lecture:

- **Characteristics of a Good Test:**
- A good test has a high probability of finding an error
- A good test is not redundant.
- A good test is neither too simple nor too complex
- A good test should be "best of breed
- Testability: Software testability is simply how easily a computer program can be tested
- Characteristics of Testable Software Operable
- The better it works, the easier it is to test Observable
- Incorrect output is easily identified; internal errors are automatically detected Controllable
- The states and variables of the software can be controlled directly by the tester Decomposable
- The software is built from independent modules that can be tested independently Simple
- The less there is to test, the more quickly we can test it.
- The program should exhibit Functional simplicity
- Structural simplicity
- Code simplicity
- Stable

If the changes are less, then the disruptions to testing are also less

- Changes to the software during testing are infrequent and do not invalidate existing tests

Understandable

- The more information exists, the smarter to test
- The architectural design is well understood; documentation is available and organized

Video Content / Details of website for further learning (if any):

https://softwaretestingfundamentals.com/

https://www.youtube.com/watch?v=8Dx_CxjoG0M

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:482

31/3/2021

Course Teacher

31/3

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu
Estd. 2000

L-29

LECTURE HANDOUTS

CSE | **II/IV**

Course Name with Code : **Software Engineering /19CSC12**
Course Teacher : **V.Karuppuchamy**
Unit : **IV-Testing And Implementation** **Date of Lecture :** 2/4/2021

Topic of Lecture : Internal and external views of Testing-white box testing

Introduction : (Maximum 5 sentences) :

Knowing the specified function that has to be performed, if we test to check whether the function is fully operational and error free then it is termed as Black box testing. Black box testing will not test the internal logical structure of the software.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Testing
- Deriving the Basis Set and Test Cases
- Condition Testing
- Testing of Simple Loops

Detailed content of the Lecture:

Black-box testing (External view of testing) Knowing the specified function that has to be performed, if we test to check whether the function is fully operational and error free then it is termed as Black box testing. Black box testing will not test the internal logical structure of the software White-box testing (Internal view of testing) White box testing checks whether the internal operations perform according to the specification This is also referred as glass box testing This involves tests that concentrate on close examination of procedural detail Logical paths are also tested. This uses the control structure of component design to derive the test cases The test cases Guarantee that all independent paths.

Video Content / Details of website for further learning (if any):

https://softwaretestingfundamentals.com/

https://www.youtube.com/watch?v=8Dx_CxjoG0M

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:494

11/4/21
Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-30

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **IV-Testing And Implementation** **Date of Lecture :** 5/4/2021

Topic of Lecture : Basis Path Testing

Introduction : (Maximum 5 sentences) :

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- white-box testing
- Block box testing
- Structure testing
- complexity

Detailed content of the Lecture:

Basis path testing is a one of the White-box testing technique.

This helps to derive a logical complexity measure of a procedural design

T(.cases derived to exercise the basis set should execute every statement in the program

At least one time during testing

Video Content / Details of website for further learning (if any):

https://softwaretestingfundamentals.com/

https://www.youtube.com/watch?v=8Dx_CxjoG0M

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:521

24/4/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu
Estd. 2000
IQAC

L-31

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**
Course Teacher : **V.Karuppuchamy**
Unit : **IV-Testing And Implementation** **Date of Lecture :** 7/4/2021

Topic of Lecture : Control Structure Testing

Introduction : (Maximum 5 sentences) :

Condition testing is a test-case design method used to check the logical conditions in the Module. This tests each condition in the program to ensure that its error- free.
A simple condition is a Boolean variable or a relational expression, possibly preceded with one NOT (¬) operator.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

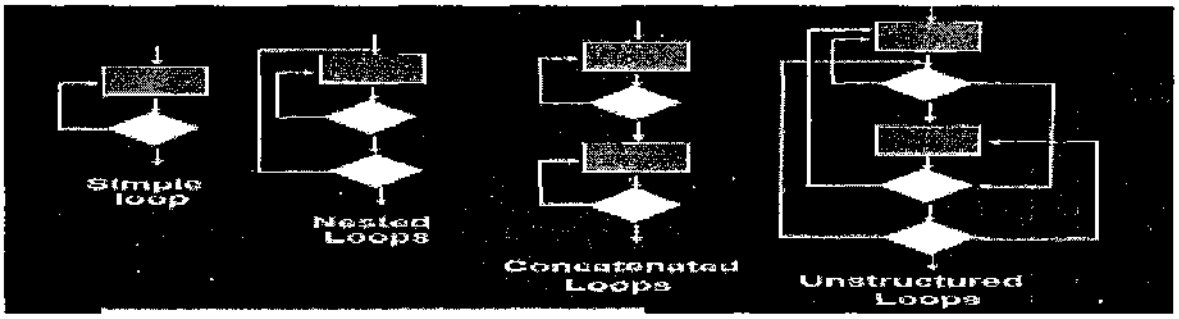
- Moving Features Between Objects
- Preparatory Refactoring
- Refactor first before adding any new features
- Plan your refactoring project and timeline carefully

Detailed content of the Lecture:

- A relational expression takes the form E1 E2 where E1 and E2 are arithmetic expressions and is one of the following: ,= , < = etc A compound condition is composed of two or more simple conditions, Boolean operators,¬ and parentheses. Boolean operators allowed in a compound condition include OR (||), AND (&), and NOT (¬).
- A condition without relational expressions is referred to as a Boolean expression.
- If a condition is incorrect, then at least one component of the condition is incorrect.
- Types of errors in a condition include Boolean operator errors, Boolean variable errors,
- Boolean Parenthesis errors, relational operator errors, and arithmetic expression errors.
- Data Flow Testing The data flow testing method selects test paths of a program according to the locations
- Definitions and uses of variables in the program. Assume that each statement in a program is assigned a unique statement number and that each function does not modify its parameters or global variables. For a statement with S as its statement number,
- DEF(S) = {X | statement S contains a definition of X} USE(S) = {X | statement S contains a use of X}

of statement S. A definition-use (DU) chain of variable X is of the form [X, S, S`], where S and S` statement numbers, X is in DEF(S) and USE(S`), and the definition of X in statement S is live at statement S'

- Every DU chain must be covered at least once. This strategy as the DU testing strategy.
- Loop Testing Loop testing is a white-box testing technique that is used to test the validity of loop constructs.
- Four different classes of loops can be defined: simple loops, concatenated loops, nested
- loops, and unstructured loops Testing occurs by varying the loop boundary values



- Testing of Simple Loops Testing of Simple Loops where n is the maximum number of allowable passes through the loop.
- 1. Skip the loop entirely
- 2. Only one pass through the loop
- 3. Two passes through the loop
- 4. m passes through the loop, where m < n
- 5. n –1, n, n + 1 passes through the loop

Testing of Nested Loops

1) Start at the innermost loop; set all other loops to minimum values

2) Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration parameter values; add other tests for out-of-range or excluded values

3) Continue until all loops have been tested

Video Content / Details of website for further learning (if any):

https://softwaretestingfundamentals.com/

https://www.youtube.com/watch?v=8Dx_CxjoG0M

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:412

5/4/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu
Estd. 2000
IQAC

L-32

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**
Course Teacher : **V.Karuppuchamy**
Unit : **IV-Testing And Implementation** **Date of Lecture :** 8/4/2021

Topic of Lecture : Black Box Testing

Introduction : (Maximum 5 sentences) :

Black-box testing is also called as behavioral testing
This testing focuses on the functional requirements of the software and the information domain of the software

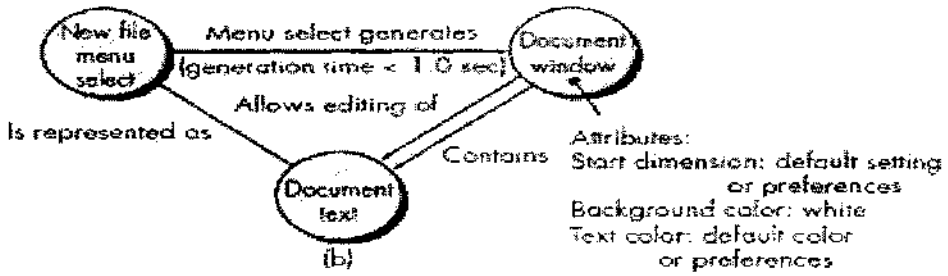
Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Incorrect or missing functions,
- Interface errors,
- Errors in data structures or external database access,
- Behavior or performance errors, and

Detailed content of the Lecture:

- Graph-Based Testing Methods The first step in black-box testing is to understand the objects and their relationships.
- Then a series of tests are defined to verify whether all the objects have the expected relationship with each other.



Nodes represent the objects and links represent the relationships between objects,

Node weights describe the properties of the node and link weights describes the characteristic of the link.

Behavioral testing methods that can make use of graphs:

1. Transaction flow modeling. The nodes represent steps in some transaction and the links represent the logical connection between steps. The data flow diagram can be used to assist in creating graphs of this type.

2. Finite state modeling. The nodes represent different user-observable states of the software and the links represent the transitions that occur to move from state to state. The state diagram can be used to assist in creating graphs of this type.

3. Data flow modeling. The nodes are data objects, and the links are the transformations that occur to translate one data object into another.

4. Timing modeling. The nodes are program objects, and the links are the sequential connections between those objects. Link weights are used to specify the required execution times.

Equivalence Partitioning

1. Equivalence partitioning is a black-box testing method which divides the input domain into classes of data and then derives the test cases from it.
2. An ideal test case single-handedly uncovers a complete class of errors and reduce the total number of test cases that has to be developed
3. Test case design is based on an evaluation of equivalence classes for an input condition
4. An equivalence class represents a set of valid or invalid states for input conditions
5. From each equivalence class, test cases are selected so that the largest number of attributes of an equivalence class are exercise at once

Video Content / Details of website for further learning (if any):

https://softwaretestingfundamentals.com/

https://www.youtube.com/watch?v=8Dx_CxjoG0M

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:523

7/4/2021

Course Teacher

Verified by HOD

L-33

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **IV-Testing And Implementation** **Date of Lecture :** 9/4/2021

Topic of Lecture : Regression Testing – Unit Testing

Introduction : (Maximum 5 sentences) :

Legression Testing is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Retest All
- Regression Test Selection
- Prioritization of Test Cases
- Selecting test cases for regression testing

Detailed content of the Lecture:

Unit testing focuses testing each individual module separately.¬ This concentrates on the internal processing logic and data structures¬ Unit testing is simplified when a module is designed with high cohesion¬ o Reduces the number of test cases o Allows errors to be more easily predicted and uncovered Unit testing concentrates on critical modules and those with high cyclomatic complexity¬ when testing resources are limited.

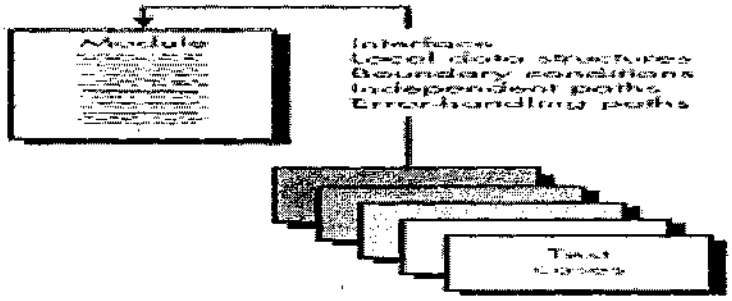


Fig: Unit test

Unit test considerations Module interface

This is tested to ensure that information flows properly into and out of the module Local data structures This ensure that data stored temporarily maintains its integrity during all steps in an algorithm execution

Boundary condition. This ensures that the module operates properly at boundary values

Independent paths Paths are exercised to ensure that all statements in a module have been executed at least once Error handling paths

Ensure that the algorithms respond correctly to specific error conditions Common Errors in Execution Paths Misunderstood or incorrect arithmetic precedence Mixed mode operations (e.g., int, float, char)

Incorrect initialization of values Precision inaccuracy and round-off error Incorrect symbolic representation of an expression (int vs. float)

Unit Test Procedures Driver

A simple main program that accepts test case data, passes such data to the component being tested, and prints the returned results Stubs Stubs serve to replace modules that are subordinate to the component to be tested A stub is a "dummy subprogram" that uses the subordinate module's interface, does minimal data manipulation, prints verification of entry, and returns control to the module undergoing testing. Drivers and stubs both represent overhead¬ – Both must be written but don't constitute part of the installed software produce.

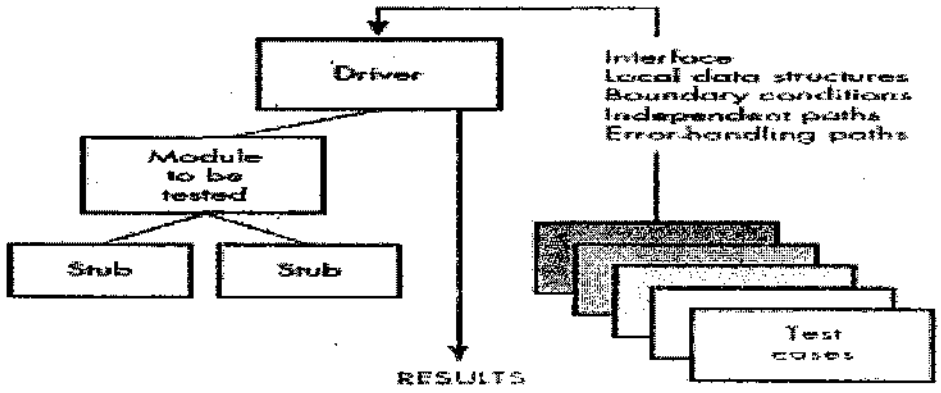


Fig: Unit Test Environment

Video Content / Details of website for further learning (if any):

https://softwaretestingfundamentals.com/

https://www.youtube.com/watch?v=8Dx_CxjoG0M

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:517

8/4/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-34

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **IV-Testing And Implementation** **Date of Lecture :** 10/4/2021

Topic of Lecture : Integration Testing – Validation Testing

Introduction : (Maximum 5 sentences) :

Integration testing is defined as a systematic technique for constructing the software Architecture and also conduct tests to uncover errors associated with interfaces.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Unit Testing
- Integration testing
- System Testing
- User Acceptance Testing

Detailed content of the Lecture:

Integration testing is defined as a systematic technique for constructing the software architecture and also conduct tests to uncover errors associated with interfaces The Objective of this testing is to take unit tested modules and build a program structure based on the prescribed design.

Two Approaches used

1. Non-incremental Integration Testing or Big Bang Approach

2. Incremental Integration Testing

1. Big Bang Approach In this approach all the components are combined in advance and the entire program is tested as a whole This is not appropriate and results in Chaos

Many seemingly-unrelated errors are encountered in this testing

Correction is difficult because isolation of causes is complicated

Once a set of errors are corrected, more errors occur, and testing appears to enter an endless loop 2.

Incremental Integration Testing Three kinds

Top-down integration

Bottom-up integration

Sandwich integration In this approach the program is constructed and tested in small increments

Errors are easier to isolate and correct

interfaces are more likely to be tested completely

A systematic test approach is applied

Top-down Integration Modules are integrated by moving downward through the control hierarchy, Beginning with the main module Subordinate modules are incorporated in either a depth-first or breadth-first fashion.

Depth First: All modules on a major control path are integrated

Breadth First: All modules directly subordinate at each level are integrated

Advantages This approach verifies major control or decision points early in the test process

Disadvantages Stubs need to be created to substitute for modules that have not been built or tested yet; this code is later discarded Because stubs are used to replace lower level modules, no significant data flow can occur until much later in the integration/testing process Steps followed in top-down integration:

1. The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.
2. Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components.
3. Tests are conducted as each component is integrated.
4. On completion of each set of tests, another stub is replaced with the real component. 5. Regression testing may be conducted to ensure that new errors have not been introduced.

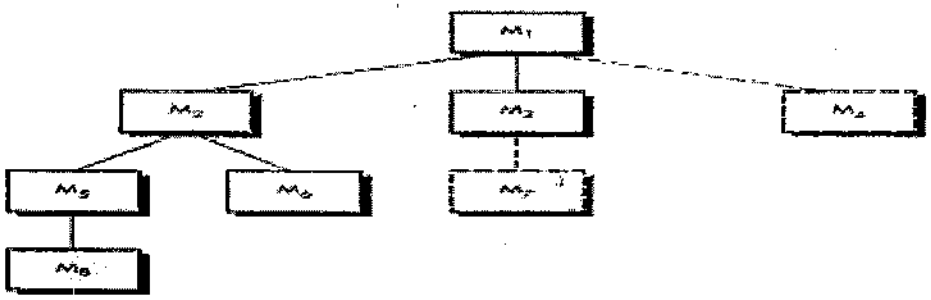


Fig: Top-down integration

Video Content / Details of website for further learning (if any):

https://softwaretestingfundamentals.com/

https://www.youtube.com/watch?v=8Dx_CxjoG0M

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:519

9/11/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu
Estd. 2000
IQAC

L-35

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**
Course Teacher : **V.Karuppuchamy**
Unit : **IV-Testing And Implementation** **Date of Lecture :** 12/4/2021

Topic of Lecture : System Testing and Debugging

Introduction : (Maximum 5 sentences) :

This is to tests for recovery from system faults Recover testing forces the software to fail in a variety of ways ar verifies that recovery is properly performed
Tests reinitialization, check pointing mechanisms, data recovery, and restart for correctness

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Deployment Testing
- Performance testing
- Debugging Process
- Debugging Strategies

Detailed content of the Lecture:

- **Security testing**
- This verifies whether the protection mechanisms built in the system will protect it from improper access
- During security testing, the tester plays the role of the hacker, attempts to acquire passwords, may purposely cause system errors, may browse through insecure data, hoping to find the key to system entry.
- Stress testing
- Stress tests are designed to confront programs with abnormal situations.
- This executes a system in a manner that demands resources in abnormal quantity, frequency, or volume
- Performance testing
- Tests the run-time performance of software within the context of an integrated system Performance testing is often combined with stress testing and usually requires both hardware and software instrumentation
- This testing can uncover situations that lead to degradation and possible system failure.

Deployment Testing

- Deployment testing is also called as configuration testing Software should be able to execute on a variety of platforms and under more than one operating system environment.
- Deployment testing exercises the software in each environment in which it is to operate.
- Deployment testing examines all installation procedures and specialized installation software that will be used by customers, and all documentation that will be used to introduce the software to end users.
- **Debugging Process**
- Debugging is the process of removing the defects.
- It occurs as a consequence of successful testing.
- The debugging process begins with the execution of a test case, assess the result and then the difference between expected and actual performance is encountered The debugging process matches the symptom with the cause and leads to error correction.
- **The debugging process will usually have one of two outcomes:**
- (1) The cause will be found and corrected
- (2) The cause will not be found. In this case, the debugger may suspect a cause, design a test case to help validate that suspicion, and work toward error correction in an iterative fashion.
- **Difficulties in Debugging**
- The symptom may disappear when another error is corrected
- The symptom may be caused by human error that cannot be easily traced
- The symptom may be a result of timing problems, rather than processing problems
- It may be difficult to accurately reproduce input conditions The symptom may be due to causes that are distributed across a number of tasks running on different processes

Video Content / Details of website for further learning (if any):

https://softwaretestingfundamentals.com/

https://www.youtube.com/watch?v=8Dx_CxjoG0M

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:518

10/4/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu
Estd. 2000
IQAC

L-36

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**
Course Teacher : **V.Karuppuchamy**
Unit : **IV-Testing And Implementation** **Date of Lecture :** 14/4/2021

Topic of Lecture : Software Implementation Techniques: Coding practices-Refactoring

Introduction : (Maximum 5 sentences) :

Code refactoring is defined as the process of restructuring computer code without changing or adding to its external behavior and functionality. There are many ways to go about refactoring, but it most often comprises applying a series of standardized, basic actions, sometimes known as micro-refactoring.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Red-Green-Refactor
- Refactoring by Abstraction
- Composing Method
- Simplifying Methods

Detailed content of the Lecture:

You need to perform code refactoring in small steps. Make tiny changes in your program, each of the small changes makes your code slightly better and leaves the application in a working state.

Run the test TDD and CI after making small changes in the refactoring process. Without running these tests, you create a risk of introducing bugs.

Do not create any new features or functionality during the refactoring process. You should refactor the code before adding any updates or new features in your existing code.

Refactoring process can affect the testing outcomes so it's good to get your QA and testing team involved in the refactoring process.

You need to accept that you won't be fully satisfied with your code. Your refactored code will be outdated in near future and you'll have to refactor it again.

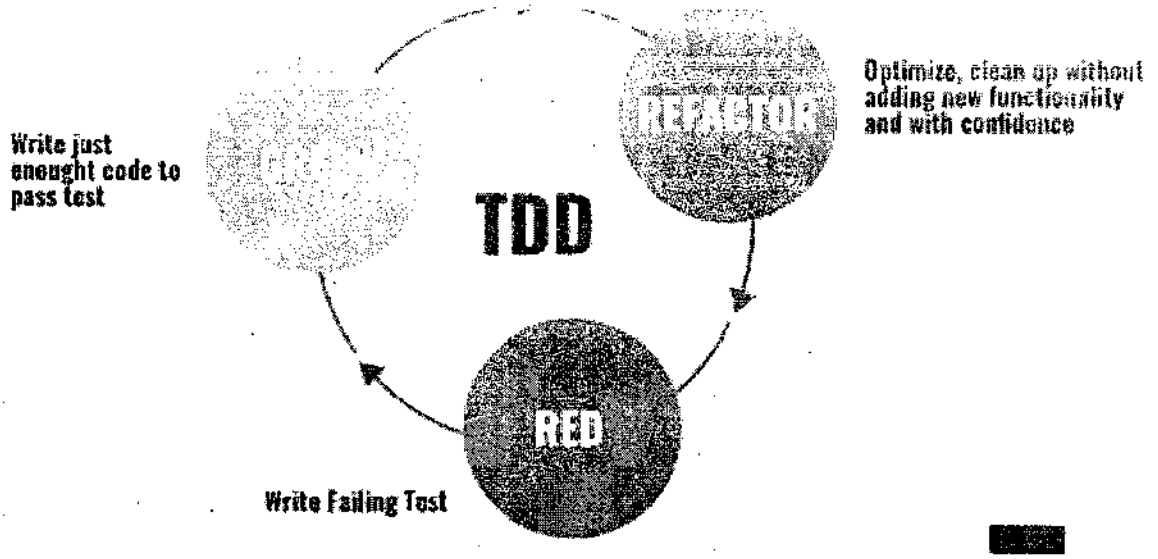
Most Common Code Refactoring Techniques

There are many approaches and techniques to refactor the code. Let's discuss some popular ones…

1. Red-Green Refactoring

Red-Green is the most popular and widely used code refactoring technique in the Agile software development process. This technique follows the "test-first" approach to design and implementation, this lays the

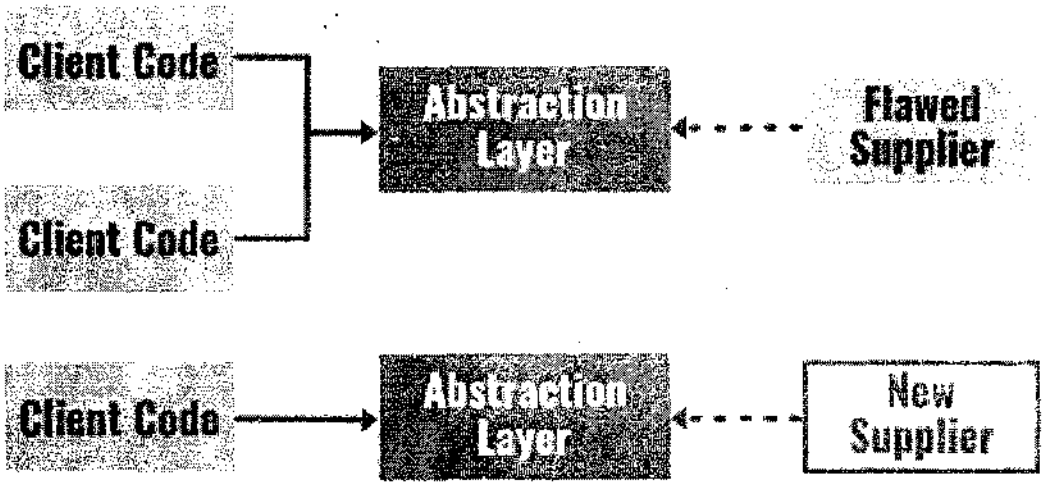
development cycle and it is performed into the three district steps.



RED: The first step starts with writing the failing "red-test". You stop and check what needs to be developed.

Green: In the second step, you write the simplest enough code and get the development pass "green" testing.

R ctor: In the final and third steps, you focus on improving and enhancing your code keeping your test green.



Video Content / Details of website for further learning (if any):

https://softwaretestingfundamentals.com/

https://www.youtube.com/watch?v=8Dx_CxjoG0M

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:521

12/4/2021
Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-37

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **V-Project Management** **Date of Lecture:** 15/4/2021

Topic of Lecture : Estimation – FP Based, LOC Based

Introduction : (Maximum 5 sentences) :

- Estimation is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable.
- Estimation determines how much money, effort, resources, and time it will take to build a specific system or product.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements Engineering**

Detailed content of the Lecture:

Estimation of the size of software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project. Various measures are used in project size estimation. Some of these are:

- Lines of Code
- Number of entities in ER diagram
- Total number of processes in detailed data flow diagram
- Function points

1. Lines of Code (LOC): As the name suggest, LOC count the total number of lines of source code in a project. The units of LOC are:

KLOC- Thousand lines of code

NLOC- Non comment lines of code

KDSI- Thousands of delivered source instruction

The size is estimated by comparing it with the existing systems of same kind. The experts use it to predict the required size of various components of software and then add them to get the total size.

Advantages:

- Universally accepted and is used in many models like COCOMO.
- Estimation is closer to developer's perspective.
- Simple to use.

Disadvantages:

- Different programming languages contains different number of lines.
- No proper industry standard exist for this technique.
- It is difficult to estimate the size using this technique in early stages of project.

Function Point Analysis : In this method, the number and type of functions supported by the software are utilized to find FPC(function point count). The steps in function point analysis are:

- Count the number of functions of each proposed type.
- Compute the Unadjusted Function Points(UFP).
- Find Total Degree of Influence(TDI).
- Compute Value Adjustment Factor(VAF).
- Find the Function Point Count(FPC).

The explanation of above points given below:

Count the number of functions of each proposed type: Find the number of functions belonging to the following types:

External Inputs : Functions related to data entering the system.

External outputs : Functions related to data exiting the system.

External Inquiries: They leads to data retrieval from system but don't change the system.

Internal Files: Logical files maintained within the system. Log files are not included here.

External interface Files: These are logical files for other applications which are used by our system.

Compute the Unadjusted Function Points(UFP) : Categorise each of the five function types as simple, average or complex based on their complexity. Multiply count of each function type with its weighting factor and find the weighted sum. The weighting factors for each type based on their complexity are as follows:

FUNCTION TYPE	SIMPLE	AVERAGE	COMPLEX
External Inputs	3	4	6
External Output	4	5	7
External Inquiries	3	4	6
Internal Logical Files	7	10	15
External Interface File	5	7	10

Find Total Degree of Influence : Use the '14 general characteristics' of a system to find the degree of influence of each of them. The sum of all 14 degrees of influences will give the TDI. The range of TDI is 0 to 70. The 14 general characteristics are: Data Communications, Distributed Data Processing, Performance, Heavily Used Configuration, Transaction Rate, On-Line Data Entry, End-user Efficiency, Online Update, Complex Processing Reusability, Installation Ease, Operational Ease, Multiple Sites and Facilitate Change.

Each of above characteristics is evaluated on a scale of 0-5.

Compute Value Adjustment Factor(VAF): Use the following formula to calculate VAF

VAF = (TDI * 0.01) + 0.65

Find the Function Point Count: Use the following formula to calculate FPC

FPC = UFP * VAF

Advantages:

- It can be easily used in the early stages of project planning.
- It is independing on the programming language.
- **It can be used to compare different projects even if they use different technologies.**

Disadvantages:

- It is not good for real time systems and embedded systems.
- Many cost estimation models like COCOMO uses LOC and hence FPC must be converted to LOC.

Video Content / Details of website for further learning (if any):

https://www.geeksforgeeks.org/software-engineering-project-size-estimation-techniques/

https://www.guru99.com/an-expert-view-on-test-estimation.html

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:702

14/9/2021
Course Teacher

14/9
Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu
Estd. 2000
IQAC

L-38

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**
Course Teacher : **V.Karuppuchamy**
Unit : **V-Project Management** **Date of Lecture:** 16/4/2021

Topic of Lecture : Make/Buy Decision ,COCOMO II

Introduction : (Maximum 5 sentences) :

- Make or buy decision is always a valid concept in business. No organization should attempt to make something by their own, when they stand the opportunity to buy the same for much less price.
- This is why most of the electronic items manufactured and software systems developed in the Asia, on behalf of the organizations in the USA and Europe.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Software Basics
- Types of Software
- Hardware Basics
- Requirements Engineering

Detailed content of the Lecture:

Four Numbers You Should Know

When you are supposed to make a make-or-buy decision, there are four numbers you need to be aware of. Your decision will be based on the values of these four numbers. Let's have a look at the numbers now. They are quite self-explanatory.

- The volume
- The fixed cost of making
- Per-unit direct cost when making
- Per-unit cost when buying

Now, there are two formulas that use the above numbers. They are 'Cost to Buy' and 'Cost to Make'. The higher value loses and the decision maker can go ahead with the less costly solution.

- people or organizations opposed to the system (negative stakeholders; see also Misuse case)
- organizations responsible for systems which interface with the system under design
- those organizations who integrate horizontally with the organization for whom the analyst is designing the system.

Risk Projection (aka Risk Estimation) Attempts to rate each risk in two ways

The probability that the risk is real

The consequences of the problems associated with the risk, should it occur.

Project planner, along with other managers and technical staff, performs four risk projection activities:

(1) establish a measure that reflects the perceived likelihood of a risk

(2) delineate the consequences of the risk

(3) estimate the impact of the risk on the project and the product

(4) note the overall accuracy of the risk projection so that there will be no misunderstandings.

Risk Mitigation, Monitoring, and Management Effective strategy must consider three issues:

- risk avoidance
- risk monitoring
- risk management and contingency planning
- Proactive approach to risk - avoidance strategy.
- Develop risk mitigation plan.

The RMMM Plan

- Risk Mitigation, Monitoring and Management Plan (RMMM) –
- Documents all work performed as Part of risk analysis and is used by the project manager as part of the overall project plan.
- Alternative to RMMM - risk information sheet (RIS)
- RIS is maintained using a database system, so that creation and information entry, priority ordering, searches, and other analysis may be accomplished easily.

Risk information sheet			
Risk ID: P02-4-32	Date: 5/9/02	Prob: 80%	Impact: high
Description: Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.			
Refinement/context: Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards. Subcondition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components. Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.			
Mitigation/monitoring: 1. Contact third party to determine conformance with design standards. 2. Press for interface standards completion; consider component structure when deciding on interface protocol. 3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.			
Management/contingency plan/trigger: *RE* computed to be $20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly. Trigger: Mitigation steps unproductive as of 7/1/02			
Current status: 5/12/02: Mitigation steps initiated.			
Originator: D. Gagne		Assigned: B. Laster	

Video Content / Details of website for further learning (if any):
http://www.sasuriceng.com/e-course-material/CSE/II-Year%20Sem%204/CS6403%20-%20SE.pdf
Important Books/Journals for further learning including the page nos.:
1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:757

19/4/2024
Course Teacher

19/4
Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-39

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **V-Project Management** **Date of Lecture:** 17/4/2021

Topic of Lecture : Planning – Project Plan

Introduction : (Maximum 5 sentences) :

- A project plan stores the outcome of project planning. It provides information about the end date, milestones, activities, and deliverables of the project.
- In addition, it describes the responsibilities of the project management team and the resources required for the project.
- It also includes the description of hardware and software (such as compilers and interfaces) and lists the methods and standards to be used. These methods and standards include algorithms, tools, review techniques, design language, programming language, and testing techniques.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Software Basics
- Types of Software
- Hardware Basics
- Requirements Engineering

Detailed content of the Lecture:

A project plan helps a project manager to understand, monitor, and control the development of software project. This plan is used as a means of communication between the users and project management team. There are various advantages associated with a project plan, some of which are listed below.

- It ensures that software is developed according to the user requirements, objectives, and scope of the project.
- It identifies the role of each project management team member involved in the project.
- It monitors the progress of the project according to the project plan.

- It determines the available resources and the activities to be performed during software development.
- It provides an overview to management about the costs of the software project, which are estimated during project planning.

Note that there are differences in the contents of two project plans depending on the kind of project and user requirements. Atypical project plan is divided into the following sections.

Introduction: Describes the objectives of the project and provides information about the constraints that affect the software project.

Project organization: Describes the responsibilities assigned to the project management team members for completing the project.

Risk analysis: Describes the risks that can possibly arise during software development as well as explains how to assess and reduce the effect of risks.

Resource requirements: Specifies the hardware and software required to carry out the software project. Cost estimation is done according to these resource requirements.

Work breakdown: Describes the activities into which the project is divided. It also describes the milestones and deliverables of the project activities.

Project schedule: Specifies the dependencies of activities on each other. Based on this, the time required by the project management team members to complete the project activities is estimated.

In addition to these sections, there are several plans that may be a part of or 'linked to a project plan. These plans include quality assurance plan, verification and validation plan, configuration management plan, maintenance plan, and staffing plan.

Quality Assurance Plan

The quality assurance plan describes the strategies and methods that are to be followed to accomplish the following objectives.

- Ensure that the project is managed, developed, and implemented in an organized way.
- Ensure that project deliverables are of acceptable quality before they are delivered to the user.

Verification and Validation Plan

The verification and validation plan describes the approach, resources and schedule used for system validation. The verification and validation plan, which comprises the following sections.

General information: Provides description of the purpose, scope, system overview, project references, acronyms and abbreviations, and points of contact. Purpose describes the procedure to verify and validate the

components of the system. Scope provides information about the procedures to verify and validate as they relate to the project. System overview provides information about the organization responsible for the project and other information such as system name, system category, operational status of the system, and system environment. Project references provide the list of references used for the preparation of the verification and validation plan. Acronyms and abbreviations provide a list of terms used in the document. Points of contact provide information to users when they require assistance from organization for problems such as troubleshooting and so on.

Reviews and walkthroughs: Provides information about the schedule and procedures. Schedule describes the end date of milestones of the project. Procedures describe the tasks associated with reviews and walkthroughs. Each team member reviews the document for errors and consistency with the project requirements. For walkthroughs, the project management team checks the project for correctness according to software requirements specification (SRS).

System test plan and procedures: Provides information about the system test strategy, database integration, and platform system integration. System test strategy provides an overview of the components required for integration of the database and ensures that the application runs on at least two specific platforms. Database integration procedure describes how database is connected to the Graphical User Interface (GUI).Platform system integration procedure is performed on different operating systems to test the platform.

Acceptance test and preparation for delivery: Provides information about procedure, acceptance criteria, and installation procedure. Procedure describes how acceptance testing is to be performed on the software to verify its usability as required. Acceptance criteria describes that software will be accepted only if all the components, features and functions are tested including the system integration testing. In addition, acceptance criteria checks whether the software accomplishes user expectations such as its ability to operate on several platforms. Installation procedure describes the steps of how to install the software according to the operating system being used.

Configuration Management Plan

The configuration management plan defines the process, which is used for making changes to the project scope. Generally, the configuration management plan is concerned with redefining the existing objectives of the project and deliverables (software products that are delivered to the user after completion of software development).

Maintenance Plan

The maintenance plan specifies the resources and processes required for making the software operational after its installation. Sometimes, the project management team (or software development team) does not carry out the task of maintenance. In such a case, a separate team known as software maintenance team performs the task of software maintenance.

Staffing Plan

The staffing plan describes the number of individuals required for a project. It includes selecting and assigning tasks to the project management team members. It provides information about appropriate skills required to perform the tasks to produce the project deliverables and manage the project. In addition, it provides information of resources such as tools, equipment, and processes used by the project management team.

Video Content / Details of website for further learning (if any):

https://www.youtube.com/watch?v=X6CkWPjLkhg

https://www.javatpoint.com/software-project-planning

https://www.youtube.com/watch?v=X6CkWPjLkhg

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:219

16/4/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-40

LECTURE HANDOUTS

IT

III/II

Course Name with Code : SOFTWARE ENGINEERING - 19ITC05

Course Teacher : E.Hemalatha

Unit : V –Project Management

Date of Lecture : 19/4/2021

Topic of Lecture : Planning Process, RFP

Introduction : (Maximum 5 sentences) :

- The project planning process involves a set of interrelated activities followed in an orderly manner to implement user requirements in software and includes the description of a series of project planning activities and individual(s) responsible for performing these activities.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements Engineering**

Detailed content of the Lecture:

Project Planning Process

Objectives and scope of the project

- ❑ Techniques used to perform project planning
- ❑ Effort (in time) of individuals involved in project
- ❑ Project schedule and milestones
- ❑ Resources required for the project
- ❑ Risks associated with the project.

Project planning process comprises several activities, which are essential for carrying out a project systematically. These activities refer to the series of tasks performed over a period of time for developing the

software. These activities include estimation of time, effort, and resources required and risks associated with the project.

Project planning process consists of the following activities.

Identification of project requirements: Before starting a project, it is essential to identify the project requirements as identification of project requirements helps in performing the activities in a systematic manner. These requirements comprise information such as project scope, data and functionality required in the software, and roles of the project management team members.

Identification of cost estimates: Along with the estimation of effort and time, it is necessary to estimate the cost that is to be incurred on a project. The cost estimation includes the cost of hardware, network connections, and the cost required for the maintenance of hardware components. In addition, cost is estimated for the individuals involved in the project.

Identification of risks: Risks are unexpected events that have an adverse effect on the project. Software project involves several risks (like technical risks and business risks) that affect the project schedule and increase the cost of the project. Identifying risks before a project begins helps in understanding their probable extent of impact on the project.

Identification of critical success factors: For making a project successful, critical success factors are followed. These factors refer to the conditions that ensure greater chances of success of a project. Generally, these factors include support from management, appropriate budget, appropriate schedule, and skilled software engineers.

Preparation of project charter: A project charter provides a brief description of the project scope, quality, time, cost, and resource constraints as described during project planning. It is prepared by the management for approval from the sponsor of the project.

Preparation of project plan: A project plan provides information about the resources that are available for the project, individuals involved in the project, and the schedule according to which the project is to be carried out.

Commencement of the project: Once the project planning is complete and resources are assigned to team members, the software project commences.

Project Plan

A project plan helps a project manager to understand, monitor, and control the development of software

project. This plan is used as a means of communication between the users and project management team. There are various advantages associated with a project plan, some of which are listed below.

- ❑ It ensures that software is developed according to the user requirements, objectives, and scope of the project.
- ❑ It identifies the role of each project management team member involved in the project.
- ❑ It monitors the progress of the project according to the project plan.
- ❑ It determines the available resources and the activities to be performed during software development.
- ❑ It provides an overview to management about the costs of the software project, which are estimated during project planning.

An **RFP** stands for "request for proposal" and is generated as part of the bidding procedure for a product or service. The purpose of an **RFP** is to provide a structured way for companies to learn about doing business with **software development** teams.

Video Content / Details of website for further learning (if any):

tp://www.sasurieengg.com/e-course-material/CSE/II-Year%20Sem%204/CS6403%20-%20SE.pdf

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:799

17/4/2021

Course Teacher

17/4

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-41

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **V-Project Management** **Date of Lecture:** 26/4/2021

Topic of Lecture : Risk Management – Identification, Projection, RMMM

Introduction : (Maximum 5 sentences) :

- The business life cycle is the progression of a business in phases over time, and is most commonly divided into five stages: launch, growth, shake-out, maturity, and decline..
- Effective risk management means attempting to control, as much as possible, future outcomes by acting proactively rather than reactively.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Software Basics
- Types of Software
- Hardware Basics
- Requirements Engineering

Detailed content of the Lecture:

- Risk management encompasses the identification, analysis, and response to risk factors that form part of the life of a business.

Identification, Projection, RMMM

- A discussion of business uses. Stakeholders (SH) are people or organizations (legal entities such as companies, standards bodies) which have a valid interest in the system. They may be affected by it either directly or indirectly. A major new emphasis in the 1990s was a focus on the identification of stakeholders. It is increasingly recognized that stakeholders are not limited to the organization employing the analyst. Other stakeholders will include:
- anyone who operates the system (normal and maintenance operators)
- anyone who benefits from the system (functional, political, financial and social beneficiaries)
- anyone involved in purchasing or procuring the system.
- In a mass-market product organization,
- product management, marketing and sometimes sales act as surrogate consumers (mass-market customers) to guide development of the product organizations which regulate aspects of the system (financial, safety, and other regulators)

Cost to Buy (CTB) = Volume x Per-unit cost when buying

Cost to Make (CTM) = Fixed costs + (Per-unit direct cost x volume)

Reasons for Making

There are number of reasons a company would consider when it comes to making in-house. Following are a few:

- Cost concerns
- Desire to expand the manufacturing focus
- Need of direct control over the product
- Intellectual property concerns
- Quality control concerns
- Supplier unreliability
- Lack of competent suppliers
- Volume too small to get a supplier attracted
- Reduction of logistic costs (shipping etc.)
- To maintain a backup source
- Political and environment reasons
- Organizational pride

Reasons for Buying

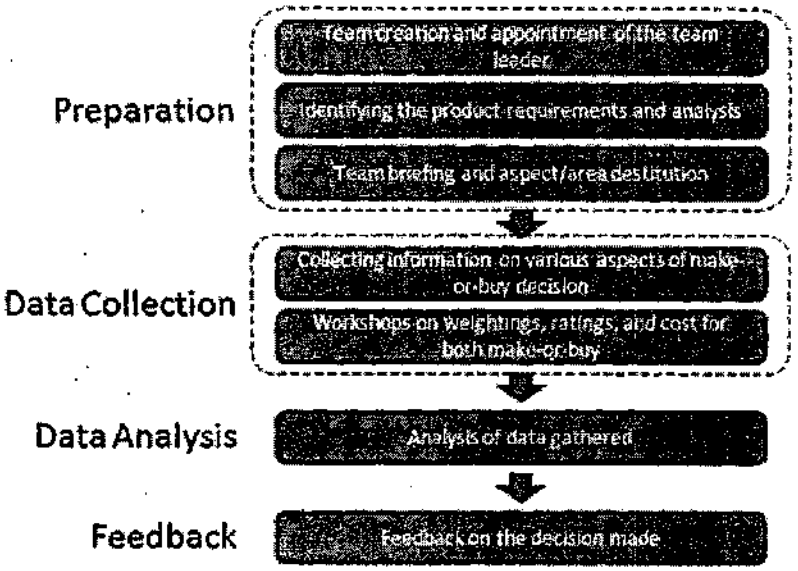
Following are some of the reasons companies may consider when it comes to buying from a supplier:

- Lack of technical experience
- Supplier's expertise on the technical areas and the domain
- Cost considerations
- Need of small volume
- Insufficient capacity to produce in-house
- Brand preferences
- Strategic partnerships

The Process

The make or buy decision can be in many scales. If the decision is small in nature and has less impact on the business, then even one person can make the decision. The person can consider the pros and cons between making and buying and finally arrive at a decision.

When it comes to larger and high impact decisions, usually organizations follow a standard method to arrive at a decision. This method can be divided into four main stages as below.



1. Preparation

- Team creation and appointment of the team leader
- Identifying the product requirements and analysis
- Team briefing and aspect/area destitution

2. Data Collection

- Collecting information on various aspects of make-or-buy decision
- Workshops on weightings, ratings, and cost for both make-or-buy

3. Data Analysis

- Analysis of data gathered

4. Feedback

- Feedback on the decision made

By following the above structured process, the organization can make an informed decision on make-or-buy. Although this is a standard process for making the make-or-buy decision, the organizations can have their own varieties.

Conclusion

Make-or-buy decision is one of the key techniques for management practice. Due to the global outsourcing, make-or-buy decision making has become popular and frequent.

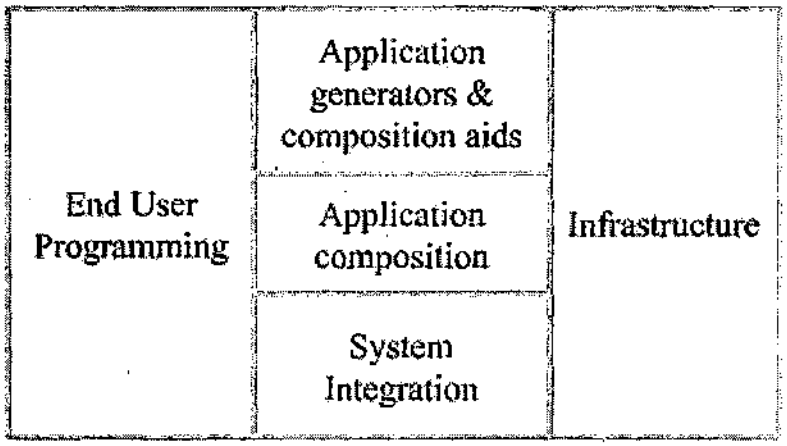
Since the manufacturing and services industries have been diversified across the globe, there are a number of

suppliers offering products and services for a fraction of the original price. This has enhanced the global product and service markets by giving the consumer the eventual advantage.

If you make a make-or-buy decision that can create a high impact, always use a process for doing that. When such a process is followed, the activities are transparent and the decisions are made for the best interest of the company.

COCOMO-II is the revised version of the original Cocomo (Constructive Cost Model) and is developed at University of Southern California. It is the model that allows one to estimate the cost, effort and schedule when planning a new software development activity.

It consists of three sub-models:

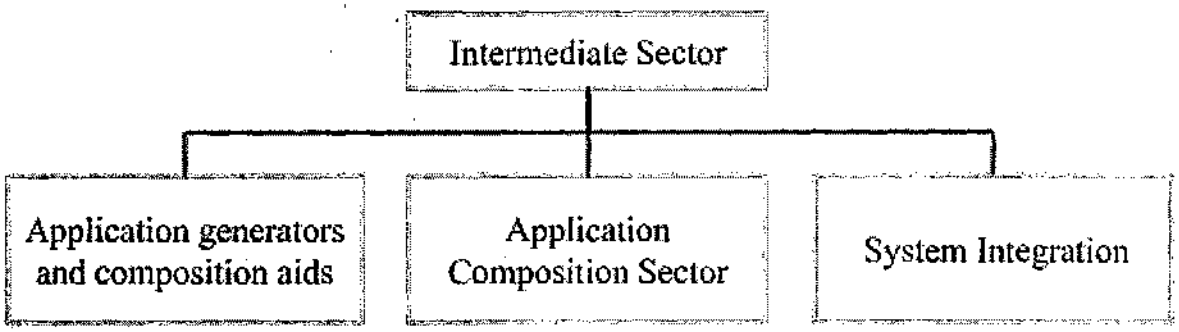


1. End User Programming:

Application generators are used in this sub-model. End user write the code by using these application generators.

Example – Spreadsheets, report generator, etc.

2. Intermediate Sector:



(a). Application Generators and Composition Aids –

This category will create largely prepackaged capabilities for user programming. Their product will have many reusable components. Typical firms operating in this sector are Microsoft, Lotus,

Oracle, IBM, Borland, Novell.

(b). Application Composition Sector –

This category is too diversified and to be handled by prepackaged solutions. It includes GUI, Databases, domain specific components such as financial, medical or industrial process control packages.

(c). System Integration –

This category deals with large scale and highly embedded systems.

3. Infrastructure Sector:

This category provides infrastructure for the software development like Operating System, Database Management System, User Interface Management System, Networking System, etc.

Stages of COCOMO II:

Stage-I:

It supports estimation of prototyping. For this it uses Application Composition Estimation Model. This model is used for the prototyping stage of application generator and system integration.

Stage-II:

It supports estimation in the early design stage of the project, when we less know about it. For this it uses Early Design Estimation Model. This model is used in early design stage of application generators, infrastructure, system integration.

Stage-III:

It supports estimation in the post architecture stage of a project. For this it uses Post Architecture Estimation Model. This model is used after the completion of the detailed architecture of application generator, infrastructure, system integration.

Video Content / Details of website for further learning (if any):

https://www.geeksforgeeks.org/software-engineering-cocomo-ii-model/

https://www.tutorialspoint.com/management_concepts/the_make_or_buy_decision.html

https://www.youtube.com/watch?v=vxZSaDtQgrs

https://www.youtube.com/watch?v=HGovKgLl10w

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:709

15/4/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-42

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **V-Project Management** **Date of Lecture:** 28/4/2021

Topic of Lecture : Scheduling and Tracking

Introduction : (Maximum 5 sentences) :

- Software project scheduling is an activity that distributes estimated effort across the planed project duration by allocating the effort to specific software engineering tasks.
- First, a macroscopic schedule is developed. ---> a detailed schedule is redefined for each entry in the macroscopic schedule.
- A schedule evolves over time.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Software Basics
- Types of Software
- Hardware Basics
- Requirements Engineering

Detailed content of the Lecture:

Basic principles guide software project scheduling:

- Compartmentalization
- Interdependency
- Time allocation
- Effort allocation
- Effort validation
- Defined responsibilities
- Defined outcomes
- Defined milestones

There is no single set of tasks that is appropriate for all projects.

An effective software process should define a collection of task sets, each designed to meet the needs of different types of projects.

A task set is a collection of software engineering work -> tasks, milestones, and deliverables.

Tasks sets are designed to accommodate different types of projects and different degrees of rigor.

Typical project types: -

- Concept Development Projects
- New Application Development Projects
- Application Enhancement Projects
- Application Maintenance Projects
- Reengineering Projects

Degree of Rigor: - Casual - Structured - Strict - Quick Reaction

Defining Adaptation Criteria: --

This is used to determine the recommended degree of rigor.

Eleven criteria are defined for software projects: -

- Size of the project
- Number of potential users
- Mission criticality
- Application longevity
- Ease of customer/developer communication
- Maturity of applicable technology
- Performance constraints
- Embedded/non-embedded characteristics
- Project staffing - Reengineering factors

Scheduling of a software project does not differ greatly from scheduling of any multitask engineering effort.

Two project scheduling methods:

- Program Evaluation and Review Technique (PERT)

- Critical Path Method (CPM)

Both methods are driven by information developed in earlier project planning activities:

- Estimates of effort
- A decomposition of product function
- The selection of the appropriate process model
- The selection of project type and task set

Both methods allow a planer to do:

- determine the critical path
- time estimation
- calculate boundary times for each task

Boundary times:

- the earliest time and latest time to begin a task
- the earliest time and latest time to complete a task
- the total float

Video Content / Details of website for further learning (if any):

https://techliebe.com/the-relationship-between-people-and-effort-in-software-engineering/

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:721

26/4/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-43

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **V-Project Management** **Date of Lecture:** 29/4/2021

Topic of Lecture : Relationship between people and effort

Introduction : (Maximum 5 sentences) :

- A comprehensive plan for a software development can be established from a software process framework. A number of a different task based milestone, work product and quality assurance points enables the frameworks activity to be adapted to characteristics of a software project.
- A software team should have extent of adaptability in choosing the software process model which is a best for a project and engineering task.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Software Basics
- Types of Software
- Hardware Basics
- Requirements Engineering

Detailed content of the Lecture:

Description of People and process management spectrum

The people:-

- People management capability maturity model (PM-CMM) has been developed by software engineering institute.
- The people management capability maturity model defines following key practice area for software people:-
- Recruiting
- Selection
- Performance
- Training
- Team / culture development
- Carrier development

It guides organizations in a creation of mature software process. The various groups are involved for the most needed communication and co-ordination issue required for any effective software. These groups can be

categorized as under:-

1) Stake holder: – Those people who are involved in software process and every software project. Stake holder can be senior manager, project manager practitioner's customer and end user.

2) Team leader: – The MOI model of leadership state characteristics that define and effective project manager motivation, organization and ideas. Successful project leaders use problem solving strategy.

3) Software team:- The people directly involved in a software project are within the software project manager scope seven project factor should be considered when planning structure of software engineering team these are as follows:-

- Difficult of problem to be solved.
- Size of resultant program.
- Time that team will stay together.
- The rigidity of a delivery date.
- The degree of communication required for project.

4) Agile teams: it is very active team which is a very small highly motivated project team consist of a formal method which result into overall development simplicity. These are all about the people description.

The Process:-

A comprehensive plan for a software development can be established from a software process framework. A number of a different task based milestone, work product and quality assurance points enables the frameworks activity to be adapted to characteristics of a software project. The frame work activity that characteristics the software processes are applicable to all software projects. The problem is to be selected process model that is appropriate for software to be a engineered by a software team.

The project manager must decide which process mode is appropriate for:

1) Customer who has a request product

2) the characteristics of product

3) the project environment.

Once the primary plan is established process decomposition begins a complete plan reflecting work task required to populate to require to framework activities must be created. These are all about process.

Process Decomposition:-

A software team should have extent of adaptability in choosing the software process model which is a best for a project and engineering task. A relatively small project might be a best accomplished using linear frequent approach. If very tight time constraints are imposed then problem can be a heavily compartmentalization RAD model. Project with other characteristics will lead to the selection of a other process model. Once a process model and process framework is decided then generic communication framework communication, planning, modeling, construction and deployment can be used. It will works for a linear model for iterative and incremental model, for evolutionary model and for concurrent or concurrent assembly models.

Process decomposition commence when the project manager asks how we accomplish the actual

activity. Process decomposition is appropriate for the small relatively and a simple project for it. It should be noted work task s must be adapted to specific need of a project. These are about the process decomposition in the software engineering for product development activities.

Video Content / Details of website for further learning (if any):

https://techliebe.com/the-relationship-between-people-and-effort-in-software-engineering/

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:725

28/4/2021

Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-44

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **V-Project Management** **Date of Lecture:** 30/4/2021

Topic of Lecture : Task Set & Network, Scheduling, EVA

Introduction : (Maximum 5 sentences) :

- Software engineering handles the relationship between people and effort management for product development phase.
- When software size is small single person can handle same project by performing steps like requirement analysis, designing, code generation, and testing etc.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Software Basics
- Types of Software
- Hardware Basics
- Requirements Engineering

Detailed content of the Lecture:

- Software engineering handles the relationship between people and effort management for product development phase.

These some points are as follows:-

1. When software size is small single person can handle same project by performing steps like requirement analysis, designing, code generation, and testing etc.

2. If the project is large additional people are required to complete. the project in stipulated in time it become easy to complete project by distributing work among people and get it done as early as possible.

3. The communication path of new comer also increase as time increase and day by day the project become extra complicated. And new customer gets confusion become more after the days by days.

4. It is possible to reduce a desire project completion date by getting more people to same point. It also possible to expand the completion date by reducing number of resources. And you can mention for the date of completion.

5. The Putnam norden Rayleigh (PNR) curve is an indication of relationship which exists between effort applied and delivery time for software project.

6. The curve indicate a minimum time value at to which indicates test cost time for delivery as use move to left to right. It is observed that curved raised non-linearly.

7. It is possible to make delivery fast; the curve rises very sharply to left of td. The pnr curve indicates that project delivery time should not be compressed much behind on td.

8. The number of delivery lines of code are also known as source statements L.

Relationship of L with effort & development time by equation can be described as L= P * E1/3 T ¾
Here 'E' represents development effort in person months, P is productivity

1. After rearranging the last equation can arrive at an expansion for development effort e. E = L3/(P3 T4) E is called as effort expanded over entire life cycle for software development and maintenance T is the development period in years.
2. And this equation is lead to E = L3 / (P3 T4) ~ 3.8 Person years. This shows that by extending last date of project with six month e.g. we can reduce the no of people from eight to four.

The outcome benefit can be gained by using less number of people over longer time to achieve the same objective. These are all about the relationship between people and effort in software engineering

Video Content / Details of website for further learning (if any):

http://www.brainkart.com/article/Relationship-between-people-and-effort,-Task-Set---Network,-Scheduling,-EVA_9100/

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:739

29/4/2024
Course Teacher

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

L-45

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Software Engineering /19CSC12**

Course Teacher : **V.Karuppuchamy**

Unit : **V-Project Management**

Date of Lecture: 31/5/2021

Topic of Lecture : Process and Project Metrics

Introduction : (Maximum 5 sentences) :

- These are metrics that pertain to Process Quality. They are used to measure the efficiency and effectiveness of various processes.
- These are metrics that relate to Project Quality. They are used to quantify defects, cost, schedule, productivity and estimation of various project resources and deliverables.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- **Software Basics**
- **Types of Software**
- **Hardware Basics**
- **Requirements Engineering**

Detailed content of the Lecture:

Project Metrics

- Schedule Variance: Any difference between the scheduled completion of an activity and the actual completion is known as Schedule Variance.
- Schedule variance = ((Actual calendar days – Planned calendar days) + Start variance)/ Planned calendar days x 100.
- Effort Variance: Difference between the planned outlined effort and the effort required to actually undertake the task is called Effort variance.
- Effort variance = (Actual Effort – Planned Effort)/ Planned Effort x 100.
- Size Variance: Difference between the estimated size of the project and the actual size of the project (normally in KLOC or FP).
- Size variance = (Actual size – Estimated size)/ Estimated size x 100.

- Requirement Stability Index: Provides visibility to the magnitude and impact of requirements changes.
- RSI = 1- ((Number of changed + Number of deleted + Number of added) / Total number of initial requirements) x100.
- Productivity (Project): Is a measure of output from a related process for a unit of input.
- Project Productivity = Actual Project Size / Actual effort expended in the project.
- Productivity (for test case preparation) = Actual number of test cases/ Actual effort expended in test case preparation.
- Productivity (for test case execution) = Actual number of test cases / actual effort expended in testing.
- Productivity (defect detection) = Actual number of defects (review + testing) / actual effort spent on (review + testing).
- Productivity (defect fixation) = actual no of defects fixed/ actual effort spent on defect fixation.
- Schedule variance for a phase: The deviation between planned and actual schedules for the phases within a project.
- Schedule variance for a phase = (Actual Calendar days for a phase – Planned calendar days for a phase + Start variance for a phase)/ (Planned calendar days for a phase) x 100
- Effort variance for a phase: The deviation between a planned and actual effort for various phases within the project.
- Effort variance for a phase = (Actual effort for a phase – a planned effort for a phase)/ (planned effort for a phase) x 100.

Process Metrics:

- Cost of quality: It is a measure of the performance of quality initiatives in an organization. It's expressed in monetary terms.
- Cost of quality = (review + testing + verification review + verification testing + QA + configuration management + measurement + training + rework review + rework testing)/ total effort x 100.
- Cost of poor quality: It is the cost of implementing imperfect processes and products.
- Cost of poor quality = rework effort/ total effort x 100.
- Defect density: It is the number of defects detected in the software during development divided by the size of the software (typically in KLOC or FP)
- Defect density for a project = Total number of defects/ project size in KLOC or FP
- Review efficiency: defined as the efficiency in harnessing/ detecting review defects in the verification stage.
- Review efficiency = (number of defects caught in review)/ total number of defects caught) x 100.

- Testing Efficiency: Testing efficiency = 1 – ((defects found in acceptance)/ total number of testing defects) x 100.
- Defect removal efficiency: Quantifies the efficiency with which defects were detected and prevented from reaching the customer.
- Defect removal efficiency = (1 – (total defects caught by customer/ total number of defects)) x 100.
- Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.

Video Content / Details of website for further learning (if any):

https://www.simplilearn.com/project-and-process-metrics-article

Important Books/Journals for further learning including the page nos.:

1. Roger S. Pressman, "Software Engineering – A Practitioner"s Approach", Seventh Edition, Mc Graw-Hill International Edition, 2010-Pg no:666

30/4/2021

Course Teacher

30/4

Verified by HOD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu.

Estd. 2000

Department of Computer Science and Engineering
Question Bank - Academic Year (2020-21)

Course Code & Course Name : **19CSC12/SOFTWARE ENGINEERING**

Name of the Faculty : **Mr.V.Karuppuchamy**

Year/Sem/Sec : **II/IV/B**

Unit-I : SOFTWARE PROCESS AND AGILE DEVELOPMENT

Part-A (2 Marks)

1. What is software engineering?
2. Define software process.
3. What are the four activities of software process?
4. Define Waterfall model.
5. Define Prototyping model.
6. Define Spiral model.
7. Write the difference between waterfall, prototype and spiral model.
8. What are the types of perspective process model?
9. Define Evolutionary model and Concurrent model.
10. What are the advantages of incremental process model?

Part-B (16 Marks)

1. Compare the following life cycle models based on their distinguishing factors, strengths and weaknesses, Waterfall Model, R Model, Spiral Model and Formal Methods Model. (Present in the form of table only use diagrams wherever necessary) Discuss waterfall model with its diagrammatic representation. (16)
2. What are the requirement issues and management issues with reference to software risk management? Also list some of the critical areas of risk? (16)
3. Explain in detail about risk projection. (16)
4. Elaborate Agile model. (16)
5. Explain the Spiral Software Process Model with neat diagram (16)
6. Narrate project scheduling in detail. (16)
7. Explain in detail about the software project management. (16)

Unit-II : REQUIREMENTS ANALYSIS AND SPECIFICATION

Part-A (2 Marks)

1. What is requirement engineering?
2. Define software requirements.
3. What are the types of requirements?
4. Define functional and non- Functional requirements.
5. Define user requirements.
6. Define system requirements and types of system requirements.
7. What are the processes in requirement engineering?
8. Define requirement validation.
9. What is classical analysis?
10. What is Structured System Analysis?

Part-B (16 Marks)

1. Explain briefly the requirement engineering process with neat sketch and describe each process with an example (16)
2. Explain the ways and means for collecting the software requirements and how are they organized and represented? (16)
3. (i) Narrate the importance of software specification of requirements. (8)
   (ii) Explain a typical SRS structure and its parts (8)
4. (i) Write short notes on data modeling? (6)
   (ii) Discuss in detail the basic structure of analysis model. (10)
5. a. Discuss in detail the Requirement elicitation with an example. (8)
   b. What is software specification? Briefly Explain it. (8)

Unit-III : SOFTWARE DESIGN

Part-A (2 Marks)

1. Define Software design.
2. What are the objectives of software design?
3. What are the levels of phases of design?
4. What is architectural design?
5. What are the issues in architectural design?
6. Define detailed design.
7. What are the concepts of software design?
8. Define refactoring?
9. What architectural styles are preferred for the following system? Why?
   (a) Networking
   (b) Web based systems

(c) Banking system

10. What is design heuristic?

Part-B (16 Marks)

1. Explain the core activities involved in User Interface design process with necessary block diagrams (16)
2. Describe in detail about architectural styles? (16)
3. Describe the golden rules for interface design (16)
4. Discuss in detail about different framework activities of User Interface Analysis and Design. (16)
5. Explain the basic concepts of software design (16)

Unit-IV : TESTING AND IMPLEMENTATION

Part-A (2 Marks)

1. What is software testing?
2. Define white box testing with diagram
3. Define basic path testing
4. How to compute the cyclomatic complexity?
5. Write any three advantages of basic path testing
6. Define control structure testing
7. What are the different types of testing in control structure testing?
8. Define loop testing with it types
9. Describe nested loop testing
10. What is black box testing?

Part-B (16 Marks)

1. Write short notes on Black Box Testing, Regression Testing and Unit Testing. (16)
2. Discuss the differences between black box and white box testing models. Discuss how these testing models may be used together to test a program schedule. (16)
3. What is black box testing? Is it necessary to perform this? Explain various test activities: (16)
4. Write short notes on
   a) Data flow testing. (8)
   b) Integration testing. (8)
5. What are the various testing strategies to software testing? Discuss them briefly: (16)

Unit-V : PROJECT MANAGEMENT

Part-A (2 Marks)

1. Define Project management
2. What is Make or buy decision?

3. List out the different approaches to size of the software.
4. What are the factors that lead to Risk?
5. Define COCOMO II Model
6. What is project planning?
7. What is RMMM?
8. What are the objectives and scope of the project planning?
9. What are the activities of Project planning process?
10. Define Project Plan

Part-B (16 Marks)

1. Define Risk. Explain the needs and activities or risk management?
2. Briefly Explain COCOMO II model.
3. Describe steps involved in project scheduling process, project timeline chart and task network.
4. Discuss on the various software cost estimation techniques
5. Write short notes on
   a) Software maintenance (8)
   b) Task scheduling with an example. (8)

17/2/21

Course Faculty **HoD**

(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC, NBA & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu.

Estd. 2000

OFFICE OF THE CONTROLLER OF EXAMINATIONS

CIA-I-SET-B

Degree & Branch	:	**B.E & CSE**	**Max Marks**	:	**50**
Year/Sem	:	**II/IV**	**Duration**	:	**1.30 Hrs**
Course Code & Course Name	:	**19CSC12/SOFTWARE ENGINEERING**	**Date**	:	**18.03.21**

19CSC12.CO1 : Apply the concepts of life cycle models to choose the appropriate model

19CSC12.CO2 : Analysis the requirements and design the software

PART-A (5x2=10Marks)
Answer all the questions

Q.No.	Questions	BT Level	Course Outcome
1.	What are the objectives of software design?	K1	CO3
2.	Define Web based systems	K1	CO3
3.	Manipulate detailed design	K3	CO4
4.	Calculate the cyclomatic complexity?	K4	CO4
5.	Compare white box testing and black box testing	K6	CO4

PART – B (1x10=10 Marks)
Compulsory Question

Q.No.	Questions	BT Level	Course Outcome
6.	Explain the core activities involved in User Interface design process with necessary block diagrams	K2	CO3

PART – C (2x15=30 Marks)
Answer all the questions

Q.No.		Questions	BT Level	Course Outcome
7.	a)	Describe the golden rules for interface design	K4	CO3
		OR		
	b)	Construct the process of Empirical model	K5	CO3
8.	a)	Explain functional and nonfunctional requirements in software engineering	K2	CO4
		OR		
	b)	Write short notes on Black Box Testing, Regression Testing and Unit Testing.	K2	CO4

Bloom's Taxonomy Level

Bloom's Taxonomy Level	K1 Remembering	K2 Understanding	K3 Applying	K4 Analyzing	K5 Evaluating	K6 Creating	Total
% of Questions	5	50	3	21	19	3	100

1. 13/3/21
2. 13/3/21

Course Faculty

13/3/21
Course Co ordinator

HoD
13/3/21

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC, NBA & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu.

Estd. 2000

OFFICE OF THE CONTROLLER OF EXAMINATIONS
CIA-I-SET-A

ANSWER KEY

Degree & Branch	:	**B.E & CSE**	**Max Marks**	:	**50**
Year/Sem	:	**II/IV**	**Duration**	:	**1.30 Hrs**
Course Code & Course Name	:	**19CSC12/SOFTWARE ENGINEERING**	**Date**	:	**25/03/2021 AN**

19CSC12.CO1 : Students will able to apply the concepts of life cycle models to choose the appropriate moc

19CSC12.CO2 : Students will able to analysis the requirements and design the software.

PART-A (5x2=10Marks)
Answer all the questions

Q.No.	Questions
1.	**Define software engineering.** Software engineering is the systematic application of engineering approaches to the development of software. A software engineer is a person who applies the principles of software engineering to design, develop, maintain, test, and evaluate computer software
2.	**Rewrite software process.** A rewrite in computer programming is the act or result of re-implementing a large portion of existing functionality without re-use of its source code. When the rewrite is not using existing code at all, it is common to speak of a rewrite from scratch.
3.	**List the four activities of software process.** The four basic process activities of **specification, development, validation and evolution** are organized differently in different development processes.
4.	**Summarize requirement engineering.** Requirements engineering is the process of eliciting stakeholder needs and desires and developing them into an agreed-upon set of detailed requirements that can serve as a basis for all subsequent development activities.
5.	**Analyze requirement validation** ✓ Completeness checks. ✓ Consistency checks. ✓ Validity checks. ✓ Realism checks. ✓ Ambiguity checks. ✓ Verifiability.

PART – B (1x10=10 Marks)
Compulsory Question

Q.No.	Questions

Time-Frame	Very Long	Long	Long	Short
Working software availability	At the end of the life-cycle	At the end of every iteration	At the end of every iteration	At the end of the life cycle
Objective	High Assurance	Rapid Development	High Assurance	Rapid development
Team size	Large Team	Not Large Team	Large Team	Small Team
Customer control over administrator	Very Low	Yes	Yes	Yes

PART – C (2x15=30 Marks)

Answer all the questions

Q.No. **Questions**

7. a) Express COCOMO model?

The constructive cost model (COCOMO) is one of the most widely used software cost estimation models. This model is developed by B.W.Boehm in 1981. COCOMO model is based on LOC, i.e., the number of lines of code. This model can be classified into three categories basic, intermediate, and detailed sub-models.

These are the essential parameter of COCOMO which is responsible for the quality of any software product:

Effort: The number of labor required to complete the work. It is measured in person-months units.

Schedule: The amount of time required to complete the work is directly proportional to the effort. It is measured in the unit of time, for example, months, and weeks.

Various models of COCOMO have been introduced to predict cost estimates at different levels. All the models are applied to different projects according to the requirements.

Software projects are classified into three categories:

Organic
Semi-detached
Embedded

Organic: In the organic type, the project deals with developing a well-understood application program; the team size is generally small. This category is for the small to medium size software product. In this type, team members have good experience and knowledge.

Semi-detached: In the semi-detached type, the essential elements are team-size, experience, knowledge of the multiple programming languages. The projects that come under the semi-detached are less familiar and hard to develop. It also requires better guidance, more experienced developers.

Embedded: In the embedded type, a software project requires the highest level of complexity, creativity, and experience. In this category, the larger team size is needed as compared to the previous models.

b) Discuss in detail the Requirement elicitation with an example

Requirements elicitation is perhaps the most difficult, most error-prone and most communication intensive software development. It can be successful only through an effective customer-developer partnership. It is needed to know what the users really need.

Requirements elicitation Activities:

Requirements elicitation includes the subsequent activities. Few of them are listed below –

Knowledge of the overall area where the systems is applied.
The details of the precise customer problem where the system are going to be applied must be understood.
Interaction of system with external requirements.
Detailed investigation of user needs.
Define the constraints for system development.

Requirements elicitation Methods:

There are a number of requirements elicitation methods. Few of them are listed below –

Interviews
Brainstorming Sessions
Facilitated Application Specification Technique (FAST)
Quality Function Deployment

1. Interviews:

Objective of conducting an interview is to understand the customer's expectations from the software. It is impossible to interview every stakeholder hence representatives from groups are selected based on their expertise and credibility.

Interviews maybe be open-ended or structured.

In open-ended interviews there is no pre-set agenda. Context free questions may be asked to understand the problem.
In structured interview, agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview.

2. Brainstorming Sessions:

It is a group technique
It is intended to generate lots of new ideas hence providing a platform to share views
A highly trained facilitator is required to handle group bias and group conflicts.
Every idea is documented so that everyone can see it.
Finally, a document is prepared which consists of the list of requirements and their priority if possible.

3. Facilitated Application Specification Technique:

It's objective is to bridge the expectation gap – difference between what the developers think they are supposed to build and what customers think they are going to get.

A team oriented approach is developed for requirements gathering.
Each attendee is asked to make a list of objects that are-

Part of the environment that surrounds the system
Produced by the system
Used by the system

Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, team is divided into smaller sub-teams to develop mini-specifications and finally a draft of specifications is written down using all the inputs from the meeting.

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University)

Rasipuram - 637 408. Namakkal Dt.

CONTINUOUS INTERNAL ASSESSMENT - 1

STATEMENT OF MARKS

NAME OF THE STUDENT : Atla Susruthi.

ROLL NO. : 1908063

SEMESTER / YEAR : IV / II

DEGREE / BRANCH : B.E / CSE

SUBJECT CODE / TITLE : 1908CI2 / Software Engineering

Name of Invigilator	Signature of Invigilator
K. SUDHA	18/3/21

Sl. No.	Date of Exam	Marks obtained (50)	Signature of the Student	Signature of the Staff
1	18.03.2021	32	25/3/21	26/3/21

HEAD OF THE DEPARTMENT

CONTROLLER OF EXAMINATIONS

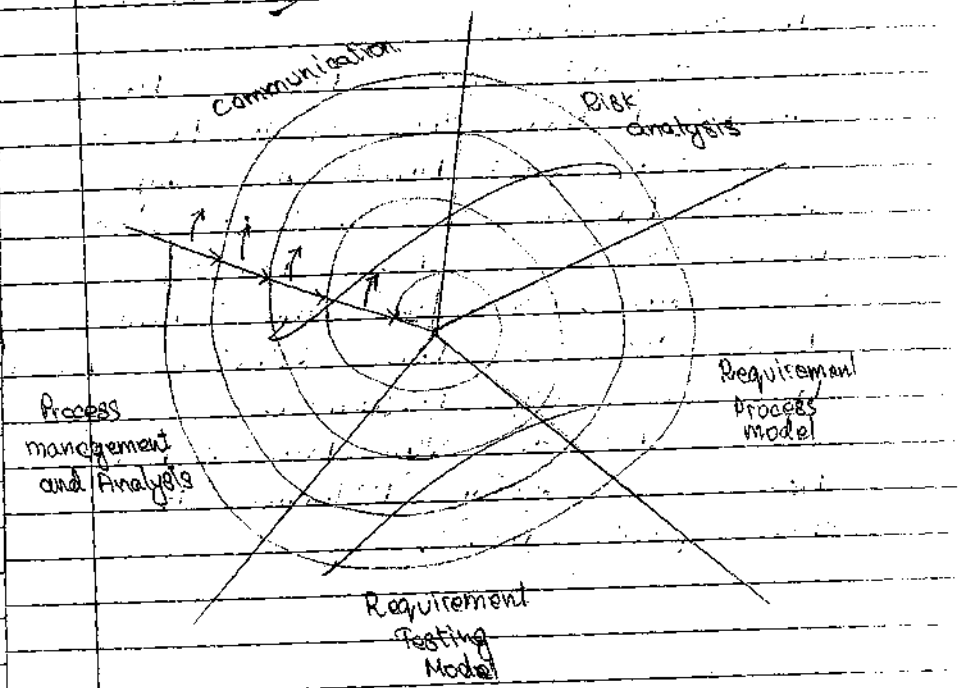
PRINCIPAL

PART-C

4. b) Spiral Software Process Model.

The Spiral Software Software Process Model are one of the life cycle of Model. The life cycle of models is have a itive Types of models.

Spiral Software model is have a some techniques. It had one diagram.



The Spiral model are spilited by the five Parts.

The Spiral model is a very hard and risky model.

In this model having a big Process and Checking and Evaluate the Process.

The Spiral model is the most Popular model. It means in this model high Priority Process are Performed In this model.

In this Spiral Process Model doesn't Perform the Small Process the Spiral Model are why doesn't Perform the Small Processes means It is a high Priority Based Model. So In this Process Model are the high Priority based Processes only Performed.

But the high Priority based Process or easily Performed in this Spiral model.

The Spiral Software Process Model are based on their distinguishing factors, Strength and weakness.

Strength of the Spiral Software Process Model.

the Spiral Software Process model are Easily Performed in this Process Model.

In this Spiral Software Process Model are very Yodifficult for Perform the high Priority Process.

In this Model have a Medium Skilled Persons have and Perform the high Priority Based Model.

Weakness of the Spiral Software Proce Model.

The Weakness of the Spiral Software Process Model are It Doesn't Perform the Low Priority Based Models.

It is Planning the Low Priority Process are very difficult to Perform In the Spiral Software Process Model are doesn't Perform the Small Processes.

The Spiral Software Process Model are have a five Part It means the Spiral Software Process model are divided by the five Simple Performance. That the five performance are Communication, Risk analysis and Process Testing or Debugging and Requirement Process model and Process Analyzing or the Evaluation this all five Process are Performed in the Spiral Software Process Model.

The Communication means the Process are Communicated with the Programmer or Analyser.

8. a). The Requirement Engineering process with the description & Sketch.

The Requirement Engineering is a Process of Analyalzing, Requirements, Testing for the Process Requirements.

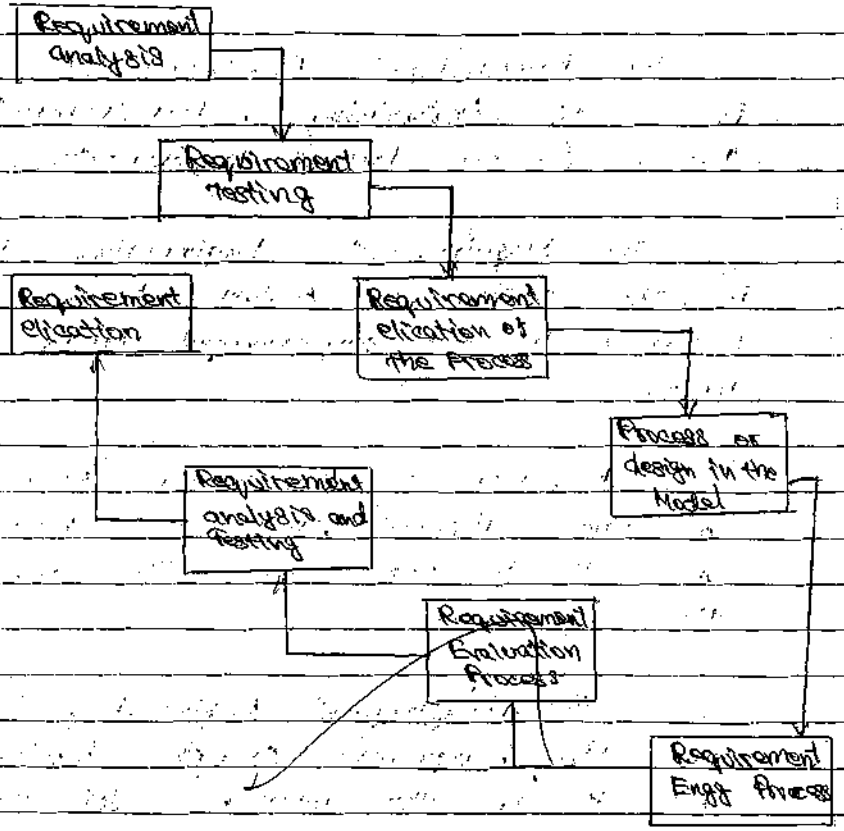
The Requirement Engineering Process is the testing or Analysis Process are Performed in the Requirement Process model.

Requirement Engineering Process are the difficult to use model and it is the used to analysis the Process

The Requirement Engineering Proc are high manpower strength having but it is the weakness for this Process.

the Requirement Software Engineering Process have a some sketch and description for the each

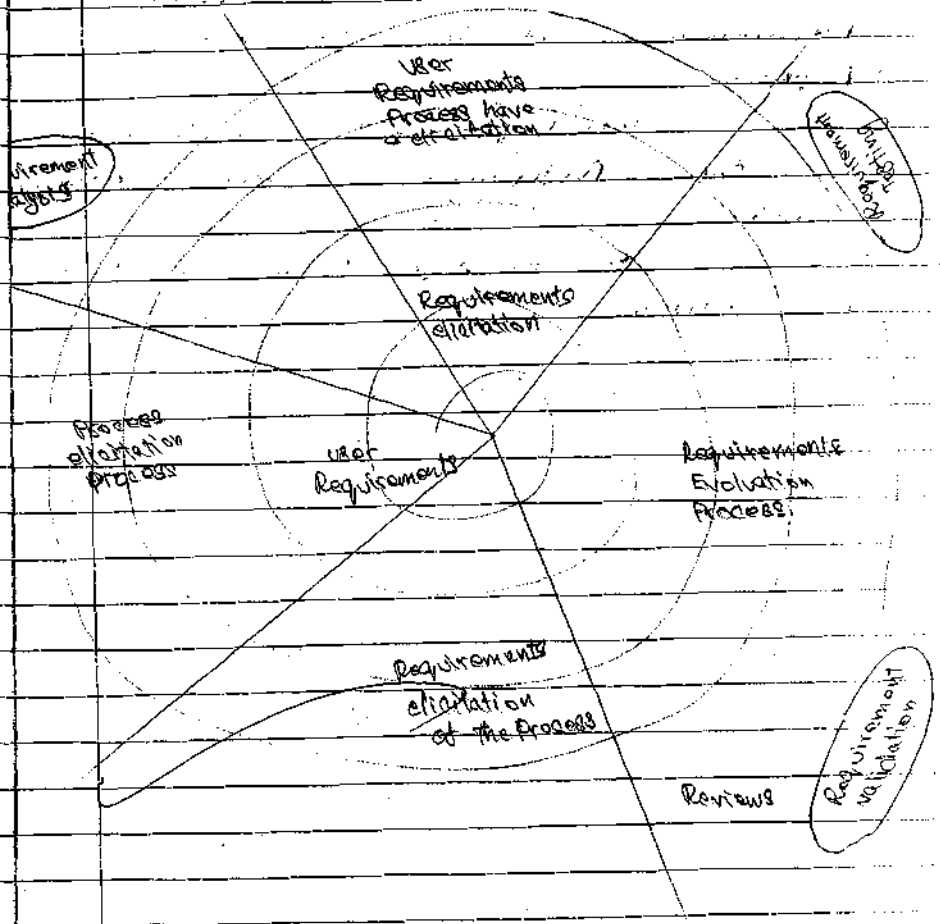
Each Process.



The Requirement engineering Process are the Process testing and Analysising and Identification and Evaluation and designing and elicitation for the

Process Requirements in this Process.

Software Requirement engineering proc



Requirement elicitation model

Requirement elicitation model are the Process of the one Iteration Performance It is the software Process.

The Requirements validations are the Performance of the Software Iteration.

the Iterative Process is the most effective

PART-B

6. The following life cycle models based on their distinguishing feature strength and weaknesses.

Waterfall model.

Waterfall model are have a Some strengths and the weakness

Strengths of the Waterfall Model.

It is very easy to Perform the Process

Waterfall Models is the very cost effective

This Model is very easy to use for the Processes.

The Waterfall model is very cost effective model.

Weakness of the Waterfall model.

The Waterfall model are difficult to Identifying the Problems.

The waterfall model are very difficult to

PART - B:

1. Software Engineering.

Software is a Program. In the Software is used to perform the any work in our System.

Software Engineering is the Efficiently software managed.

2. Software Process:

the Software Process is the Evaluating and Testing

3. Four activities of Software Process.

Communication

Identification

Planning

Testing.

MUTHAYAMMAL ENGINEERING COLLE(

(An Autonomous Institution, Affiliated to Anna University)

Rasipuram - 637 408. Namakkal Dt.

CONTINUOUS INTERNAL ASSESSMENT - I

STATEMENT OF MARKS

NAME OF THE STUDENT : DINESH. C

ROLL NO. : 19CS066

SEMESTER / YEAR : IV - II

DEGREE / BRANCH : BE - CSE

SUBJECT CODE / TITLE : 19CSC12 / SOFTWARE ENGINEERING

Name of Invigilator	Signature of Invigilator
T. Aravind	18/3/21

Sl. No.	Date of Exam	Marks obtained (50)	Signature of the Student	Signature of the Staff
1	18.3.21	45	C. Dinesh 25/3/2021	25/3/21

HEAD OF THE DEPARTMENT

CONTROLLER OF EXAMINATIONS

PRINCIPAL

Part-A

1. **Software Engineering**

Software Engineering is defined as the process to be develop a Software that is known as Software Engineering.

That the process of develop is known as Software Engineering.

2. **Software Process**

A Software is defined as a Program that execute a set of instructions for its process.

The Software Process is defined as to perform some actions do in a System by an user.

That the process is well known as Software Process.

3. **Software Process of Activities**

The Software Process has Contain four types of activities that are:

Communicating
Interfaces
Analysis & Design.

4. Requirement Engineering:

The Requirement Engineering is defined as the process to perform some

That are:

Requirement Gathering
Analysis
Documentation
Validating.

There are the activities in Requirement engineering.

Requirement Gathering:

The Requirement Gathering is defined as the process to gather an information of data for the requested Requirement

This is known as Requirement Gathering

Analysis:

An Analysis is defined as the process to find an error in a given data of information to be process.

That is known as Analysis.

There are the Requirement engineering Process:-

5. Analyze Requirement Validation:

An Analyze of requirement Validation is defined as the process to analyse an information for the data to perform an activities

That the Requirement will Gathered then next Stage of process is Analyze Requirement Validation

This is called an analyze of Requirement Validation.

Part - B.

6. Waterfall Model:

The waterfall model is one of the model in life cycle mode.

Then the waterfall mode is defined as the process to perform is sequencial to be perform.

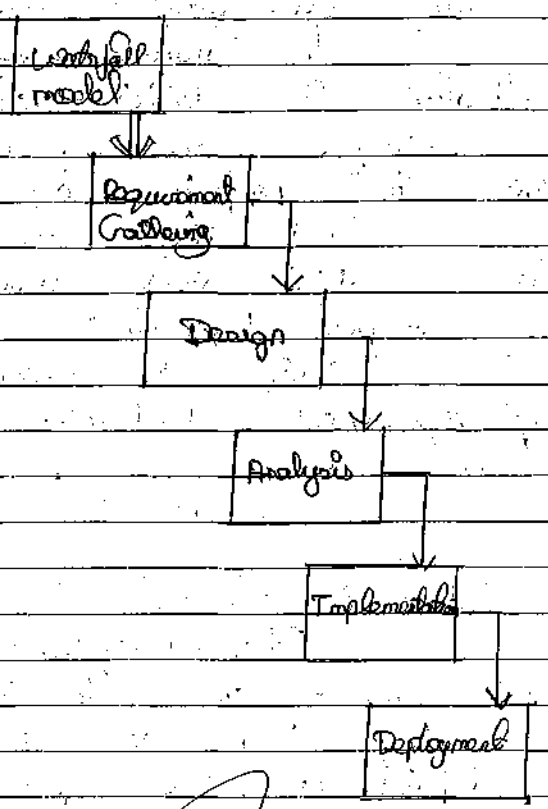
That means Step by Step process.

If it will be clear the Consumer have feel Good.

If the Consumer have Small doubts in that production.

It make again start first

Diagram:



Process:

The waterfall model is a easy process method.

It is a Sequencial process

That means Step by Step process to perform

This is the process in waterfall model.

Advantages:

Easy to Design
Easy to Understand
Cost will be low.

Disadvantage:

It doesn't backward Come's process.

Then Additional process can't to add or remove not easy.

Spiral Model

The Spiral Model is also one of the model in life cycle models.

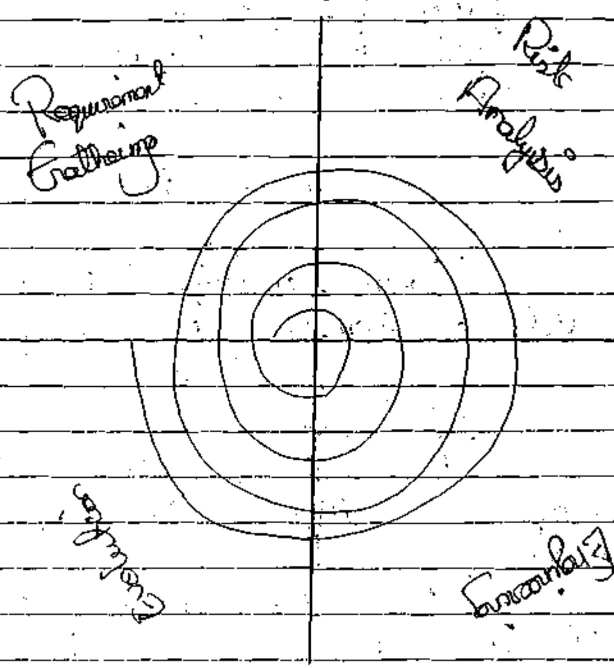
In this Spiral Model is defined as the process to (dep) design and maintained the project by the performance.

It is useful term for the project models.

Advantage:

Easy to Understanding
Easy to Add or Remove process
Low Cost
Quick Processing.

Spiral Model Diagram:



Disadvantage:

If a currency project will be not matching is weekly.

It backward will be carry, the cost efficient will be increased.

Process:

It has contain four blocks for an individual.

To be performed that particular place only.

R-model:

The full form of R-m is Relational model.

In the Relating model is defined as the process to perform a Relation-ship two entities.

That the process is known as Relational model.

It is one of the model in life Cycle Model.

Then also it has depended on before stage.

The is called R-model

Formal Method Model:

The formal method model is defined as to give the formal class for each project.

It has to perform a business purpose

That contain some activities.

That are:

ii) Analysis
i) Requirement Gathering
iii) Testing
iv) Documentation.

Part-C

7. b. The Spiral Model

The Spiral Software Process Model is defined as a process to design a project in simple steps.

It is one of the model is life Cycle Model.

Then it contains four stages that are:

Requirement Gathering
Risk Analysis
Engineering
Evaluation

Requirement Gathering

The Requirement Gathering is defined as the process to gather an information for each requirement process.

This is one of the stage in Spiral Software Process Model.

Then it is also first stage.

This is the process in Spiral Model.

In this Requirement will be Gathered then it move to the next stage of Risk Analysis for to find the process.

Risk Analysis:

The Risk Analysis is defined as the process to analyse for to finding an error in a project or no

By using the Safty mode

It is the second stage Spiral Software Process Model.

The Requirement will be gathe in Requirement Gathering stage.

It move from Requirement Ga stage into Risk Analysis stage.

This stage to find an erro in this project.

Then it has to be correct

And it moves to Engineering stage.

Engineering (Stage)

An Engineering stage is defi as the process to perform to build the unconnected statements in a single segments.

After Complete the Risk Analy it move to Engineering.

It has to build the pro for to shows for the Consumer to interact or not.

Evolution:

An Evolution is defined as the process to completed Stage in the model.

When the project will completed the Engineering process Stage.

Then it come's to an evolution Stage.

This is the final Stage in Spiral process model.

In an engineer stage will be completed.

The Consumer to be review that the project

If any criteria can interrupted from the project.

It Start will the before Stage again and again.

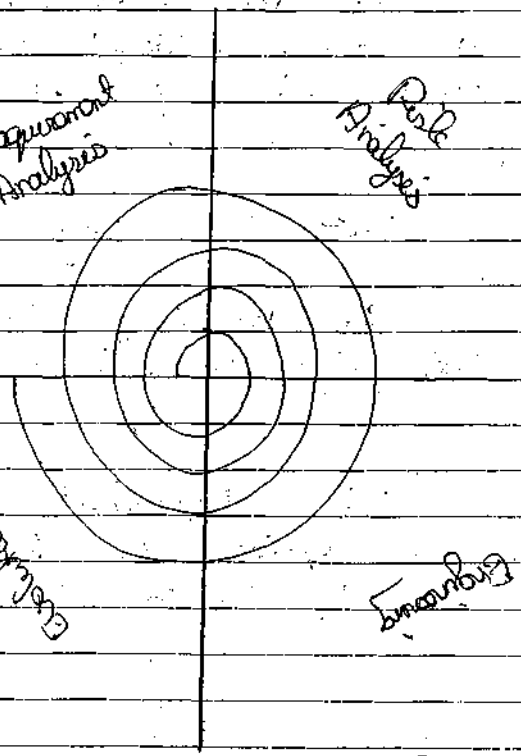
When the Consumer will be enough that the project will be good

That time has been process.

If the Consumer will be start to good criteria of the project will be done.

It has been for an approved.

Diagram:



There are the four Stage's Spiral model.

Advantages

Easy to understand.

Quick to Solution

Easy to Design.

There are an advantages in Spiral model.

Disadvantage:

In this Model has Certain based on user building Concept so an used to give the Serbes

If a Consumer will not Satisfied

That the Stage will be Re-design

If the Stage will be Re-design the Cost will be high.

8. Requirement Engineering Process

a. The Requirement Engineering Process is defined as the process to based one by one steps.

Identification

Analysis

Documentation

Sketch.

This is step of process in Requirement Engineering Process.

Each has Certain Separated working process.

To based on work will be performed on it.

It has own process.

Identification:

An Identification is defined as the process to identified the given document process.

That the process to be perf by it's an identification

If this requirement process will be process to be performed be on its works.

This is the first process Requirement engineering process.

Analysis:

An Analysis is defined as the process to analysis any mistakes on error in a project or not.

By the help of analyses.

This is also one of the Process of steps in Requirement Engineering process.

Each has Certain to perform Separated working principles.

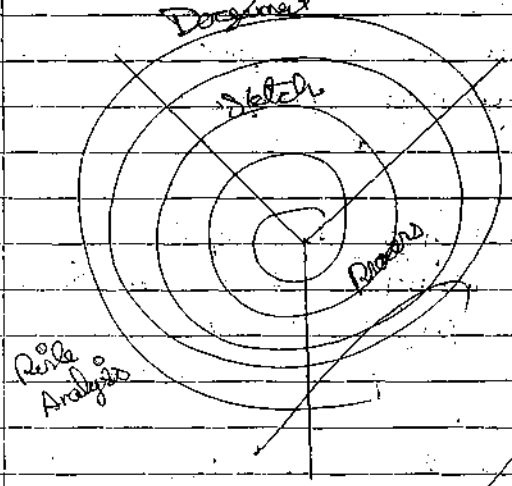
It is the most important S in the requirement engineering proces

This is also known as Risk Analysis.

Documentation

The Documentation is defined as the process to converted the Softcopy into an Hard Copy like a Documentation

This is also one of the steps in Requirement engineering process -



This is the document stage

Sketch:

To draw a sketch for designing

That the process will be executed.

In this requirement engineering process there are the process in the Requirement Engineering process.

MUTHAYAMMAL ENGINEERING COLLEG

(An Autonomous Institution, Affiliated to Anna University)

Rasipuram - 637 408. Namakkal Dt.

CONTINUOUS INTERNAL ASSESSMENT - 1

STATEMENT OF MARKS

NAME OF THE STUDENT : RITHIKA . V

ROLL NO. : 1908105

SEMESTER / YEAR : IV - II

DEGREE / BRANCH : BE - CSE

SUBJECT CODE / TITLE : 19CSC12 | SOFTWARE ENGINEERING

Name of Invigilator	Signature of Invigilator
S. Prasanna Devaraman.	18/3/24

Sl. No.	Date of Exam	Marks obtained (50)	Signature of the Student	Signature of the Staff
1	18.3.24	36	25/3/24	26/3/24

HEAD OF THE DEPARTMENT

CONTROLLER OF EXAMINATIONS

PRINCIPAL

Part-C

7. (b) Spiral Software Process :

⇒ It is a risk-handler process by t stakeholder to control a process in t model.

⇒ Spiral software process is a riskey customer, product manager and als a stackholder. So, that they always prefer to customer opinion.

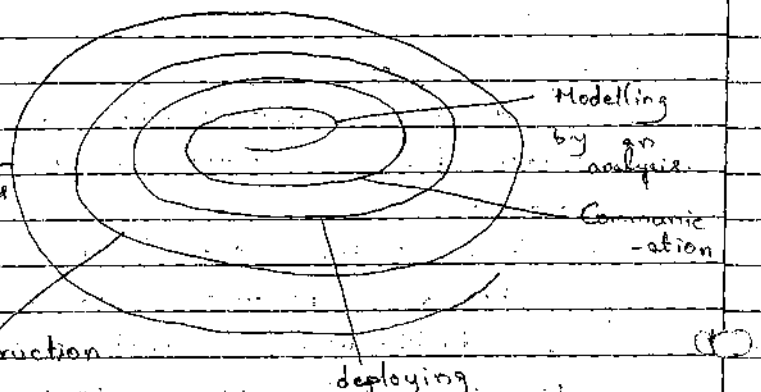
⇒ There are two methods in this process.

* It is modelling by a one spe person who control the product by product manager.

* Best seller product is specified by a stackholder.

⇒ The stackholder will give mak imp to a product and they will pr for the customer response.

⇒ the customer will receive the b quality product it means a the Oder will be so late. So, custo will get a best quality of a product.



Advantages and dis-advantages:-

* Good & bad the product is selling to a customer.
* If quality & quantity product means they have to wait many days, Because the product of order will be so late.
* These all are considered by the stackholder.
* They need a feedback of customers department.

=> The risk-handler spiral software process is layering the step by step.

Three commands:-

There are three commands: first, second, third.

=> first command is they did not mention the time for a product. If they mentioned time they did no give the quality product.

=> Second command is they sent a product will be destroyed, the pr is very normal of quality and quantity of a product, if sent product very slow then it be chaos.

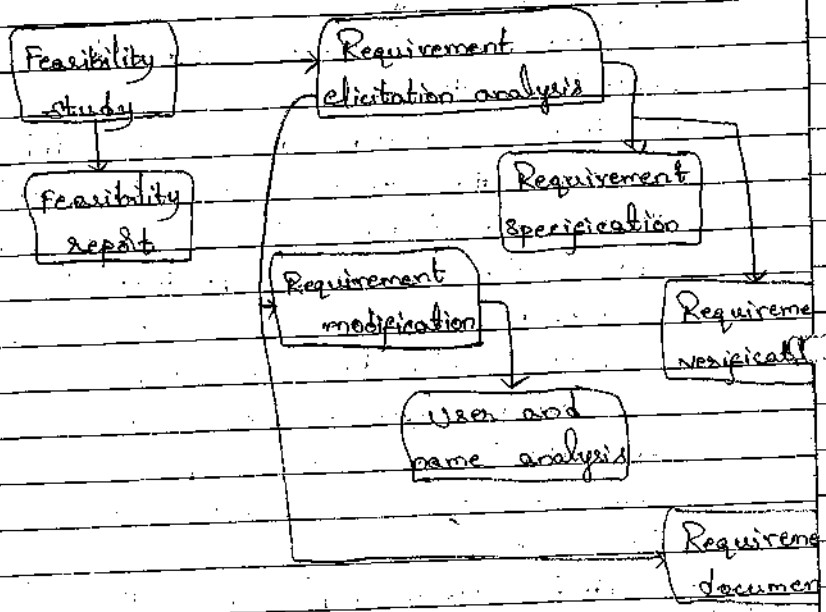
=> Third command is they must feed back from customers and consider as stackholder.

8. ⓐ Requirement Engineering process:-

=> It is a process to control by stackholder to maintain a company.

=> The requirements of a process is important to maintain the sy

=> Stack-holder will take all responsibilities of the proces & maintain the system.



⇒ Requirement elicitation is a process that they require a household process of a stack holder.

Feasibility study :-
* Create a reason.
* Acceptable, understandable.
⇒ It will create a table to modify the resources from one to one

in the sources.

Requirement elicitation analysis :-
⇒ Discuss with a right person.
⇒ Cost seen by the head.
If we discuss with a right p in a company, they conclude th product good & bad. Otherwise they only convey a message the product will be them.
We have reduce the cost by se to a customer.

Requirement specification :-
⇒ Whether the product quality is a bad.
⇒ It is a unbreakable project. Firstly, they think about t quality of the product to keep rating. After order received by t company, immediately they sent a ratings to customer. It is very important to a engineer.

Requirement verification:-

⇒ Prototyping

⇒ Product will be safe.

Prototyping means a product will be check again and again. Because it may be recycling the product.

Product will receive by safely by the company members to a customer. So they give respect to stackholder.

Document requirement:-

They finally create a document by the company. The ratings will taken from the customers and then they will rate the product.

Quantity and quantity is also based on a requirement document through given by the company members. This is a document requirement.

Part - B

6. Waterfall Model:-

Strengths and weakneses:-

⇒ The waterfall model is also as spiral software model. So that the did not have big process.

⇒ It does not have a separate variation. It will be life cycle.

⇒ Waterfall model is some what as spiral model.

R-model:-

Strengths and weakneses:-

⇒ It is different from R waterf model. Because it will manage b its own process to a company.

⇒ The stackholder will consider diff to this company.

⇒ It does not depends upon an other company.

⇒ R - Model is controlled by the it means like stack holder.

Spiral model :-

Strengths and Weaknesses :-

=> It is a risk-handler controlled by the stack holder then they will consider a certificate.

=> The customer feed back is so important in spiral model.

=> This is some what similar to water fall model.

=> It will taken a decision one to another, then it will correct the headers by spiral model.

Formal Methods model :-

=> Formal method model is completly different from all the models.

=> They will take their own decisions on a product. Either the product is selling & purchasing by the customers.

=> It will be lesser time also than we compare to all models.

=> It is a formal methods model.

Part - A

1. Software Engineering :-

Software engineering is refers to a developing a program & a system by day by day. It will develops software company in daily life. This called as a software engineering.

2. Software Process :-

The process of a computer is develop the concept & system is. It is also called as a methodology. This is one set to another set will develop. It is called as a software process.

3. The four activities of software process :-

Modelling

Planning

Construction process

deploying process.

4. Requirement Engineering :-

It requires some process & a execution system by an engineering. This is mainly developed in the

software companies. It is called as a requirement engineering.

5. Requirement validation:-
It refers to a validation of the process checking the requirement actually define the system that the customer wants. This is called as requirement validation.

3. The activities of software process:-
* Software specification
* Software validation
* Software define.
* Software implementation.

(Approved by AICTE, New Delhi, Accredited by NAAC, NBA & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu.
Estd. 2000

OFFICE OF THE CONTROLLER OF EXAMINATIONS

CIA-II-SET-A

Degree & Branch	:	**B.E & CSE**	**Max Marks**	:	**50**
Year/Sem	:	**II/IV**	**Duration**	:	**1.30 Hrs**
Course Code & Course Name	:	**19CSC12/SOFTWARE ENGINEERING**	**Date**	:	**22.04.2021**
19CSC12.CO3	:	Construct a design for a real-world problem			
19CSC12.CO4	:	Design and develop test cases.			

PART-A (5x2=10Marks)
Answer all the questions

Q.No.	Questions	BT Level	Course Outcome
1.	List the Steps involved in Software Designing process	K1	CO3
2.	State Decomposability	K1	CO3
3.	Write the difference between testing and validation	K3	CO4
4.	Define Regression Testing	K1	CO4
5.	Review Refactoring	K2	CO4

PART – B (1x10=10 Marks)
Compulsory Question

Q.No.	Questions	BT Level	Course Outcome
6.	Explain Architectural Design in Software development process	K2	CO3

PART – C (2x15=30 Marks)
Answer all the questions

Q.No.		Questions	BT Level	Course Outcome
7.	a)	Distinguish Software measurement in detail	K4	CO3
		OR		
	b)	Construct the process of Empirical model	K5	CO3
8.	a)	Explain functional and nonfunctional requirements in software engineering	K2	CO4
		OR		
	b)	Discuss the followings 1. black box testing 2.Integration Testing	K2	CO4

Bloom's Taxonomy Level

Bloom's Taxonomy Level	K1 Remembering	K2 Understanding	K3 Applying	K4 Analyzing	K5 Evaluating	K6 Creating	Total
% of Questions	8	53	3	19	19	0	100

Course Faculty 20/4/21 Course Co ordinator 20/4/21 HoD 20/4/21

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC, NBA & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu.

Estd. 2000

OFFICE OF THE CONTROLLER OF EXAMINATIONS

CIA-II-SET-A

ANSWER KEY

Degree & Branch	:	**B.E & CSE**	**Max Marks**	:	**50**
Year/Sem	:	**II/IV**	**Duration**	:	**1.30 Hrs**
Course Code & Course Name	:	**19CSC12/SOFTWARE ENGINEERING**	**Date**	:	20.05.2021

16CSD13.CO3 : Construct a design for a real-world problem

16CSD13.CO4 : Design and develop test cases.

PART-A (5x2=10Marks)

Answer all the questions

Q.No.	Questions
1.	**List the Steps involved in Software Designing process** 1. Requirements 2. Design and Prototyping 3. Software Development 4. Testing 5. Deployment. 6. Maintenance and Updates 7. Waterfall.
2.	**State Decomposability** To become broken down into components; disintegrate. 2. To decay; rot or putrefy. See Synonyms at decay.
3.	**Write the difference between testing and validation** Validation set: A set of examples used to tune the parameters of a classifier, for example to choose the number of hidden units in a neural network. – Test set: A set of examples used only to assess the performance of a fully-specified classifier. These are the recommended definitions and usages of the terms.
4.	**Define Regression Testing** Regression testing is re-running functional and non-functional tests to ensure that previously developed and tested software still performs after a change. If not, that would be called a regression
5.	**Review Refactoring** Fully Revised and Updated–Includes New Refactorings and Code Examples "Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

PART – B (1x10=10 Marks)

Compulsory Question

Q.No.	Questions
6.	Explain Architectural Design in Software development process

The different internal product attributes are size, effort, cost, specification, length, functionality, modularity, reuse, redundancy, and syntactic correctness. Among these size, effort, and cost are relatively easy to measure than the others.

The different external product attributes are usability, integrity, efficiency, testability, reusability, portability, and interoperability. These attributes describe not only the code but also the other documents that support the development effort.

Resources
These are entities required by a process activity. It can be any input for the software production. It includes personnel, materials, tools and methods.

The different internal attributes for the resources are age, price, size, speed, memory size, temperature, etc. The different external attributes are productivity, experience, quality, usability, reliability, comfort etc.

Determining Relevant Measurement Goals
A particular measurement will be useful only if it helps to understand the process or one of its resultant products. The improvement in the process or products can be performed only when the project has clearly defined goals for processes and products. A clear understanding of goals can be used to generate suggested metrics for a given project in the context of a process maturity framework.

The Goal–Question–Metric (GQM) paradigm
The GQM approach provides a framework involving the following three steps –

Listing the major goals of the development or maintenance project

Deriving the questions from each goal that must be answered to determine if the goals are being met

Decide what must be measured in order to be able to answer the questions adequately

To use GQM paradigm, first we express the overall goals of the organization. Then, we generate the questions such that the answers are known so that we can determine whether the goals are being met. Later, analyze each question in terms of what measurement we need in order to answer each question.

Typical goals are expressed in terms of productivity, quality, risk, customer satisfaction, etc. Goals and questions are to be constructed in terms of their audience.

To help generate the goals, questions, and metrics, Basili & Rombach provided a series of templates.

Purpose – To (characterize, evaluate, predict, motivate, etc.) the (process, product, model, metric, etc.) in order to understand, assess, manage, engineer, learn, improve, etc. Example: To characterize the product in order to learn it.

Perspective – Examine the (cost, effectiveness, correctness, defects, changes, product measures, etc.) from the viewpoint of the developer, manager, customer, etc. Example: Examine the defects from the viewpoint of the customer.

Environment – The environment consists of the following: process factors, people factors, problem factors, methods, tools, constraints, etc. Example: The customers of this software are those who have no knowledge about the tools.

OR

b) Construct the process of Empirical model

- ✓ Interoperability
- ✓ Environmental

These can can be classified as :

1. **Performance constraints –**
   Reliability, security, response time, etc.
2. **Operating constraints –**
   These include physical constraints (size, weight), personnel availability, skill level considerations, system accessibility for maintenance, etc.
3. **Interface constraints –**
   These describe how the system is to interface with its environment, users, and other systems. For example, user interfaces and their qualities (e.g., user-friendliness).
4. **Economic constraints –**
   Immediate and/or long-term costs.
5. **Lifecycle requirements** – Quality of the design:
   These measured in terms such as maintainability, enhance ability, portability.

Advantages of Non-Functional Requirement :

- They ensure the software system follows legal and adherence rules.
- They specify the quality attribute of the software.
- They ensure the reliability, availability, performance, and scalability of the software system
- They help in constructing the security policy of the software system.
- They ensure good user experience, ease of operating the software, and minimize the cost factor.

Disadvantages of Non-functional requirement :

- The nonfunctional requirement may affect the various high-level software subsystem.
- They generally increase the cost as they require special consideration during the software architecture/high-level design phase.
- It is difficult to change or alter non-functional requirements once you pass them to the architecture phase.

OR

b) Discuss the followings

black box testing

Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing.

Initially, the requirements and specifications of the system are examined.
Tester chooses valid inputs (positive test scenario) to check whether SUT processes them correctly. Also, some invalid inputs (negative test scenario) are chosen to verify that the SUT is able to detect them.
Tester determines expected outputs for all those inputs.
Software tester constructs test cases with the selected inputs.

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC, NBA & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu.

Estd. 2000

OFFICE OF THE CONTROLLER OF EXAMINATIONS

CIA-II-SET-A

ANSWER KEY

Degree & Branch	:	**B.E & CSE**	**Max Marks**	:	**50**
Year/Sem	:	**II/IV**	**Duration**	:	**1.30 Hrs**
Course Code & Course Name	:	**19CSC12/SOFTWARE ENGINEERING**	**Date**	:	20.05.2021

16CSD13.CO3 : Construct a design for a real-world problem

16CSD13.CO4 : Design and develop test cases.

PART-A (5x2=10Marks)

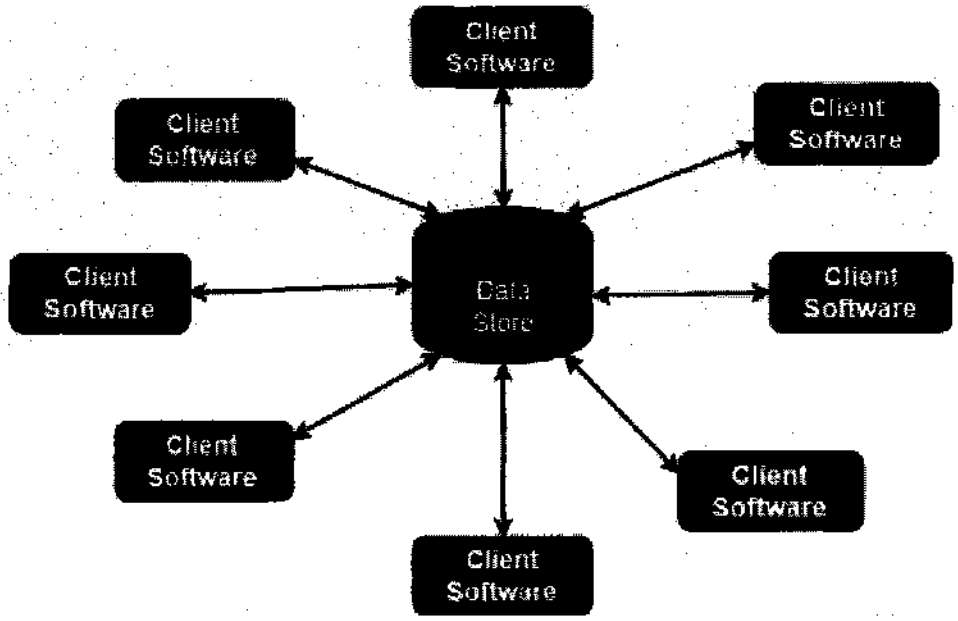
Answer all the questions

Q.No.	Questions
1.	**List the Steps involved in Software Designing process** 1. Requirements 2. Design and Prototyping 3. Software Development 4. Testing 5. Deployment. 6. Maintenance and Updates 7. Waterfall.
2.	**State Decomposability** To become broken down into components; disintegrate. 2. To decay; rot or putrefy. See Synonyms at decay.
3.	**Write the difference between testing and validation** Validation set: A set of examples used to tune the parameters of a classifier, for example to choose the number of hidden units in a neural network. – Test set: A set of examples used only to assess the performance of a fully-specified classifier. These are the recommended definitions and usages of the terms.
4.	**Define Regression Testing** Regression testing is re-running functional and non-functional tests to ensure that previously developed and tested software still performs after a change. If not, that would be called a regression
5.	**Review Refactoring** Fully Revised and Updated–Includes New Refactorings and Code Examples "Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

PART – B (1x10=10 Marks)

Compulsory Question

Q.No.	Questions
6.	Explain Architectural Design in Software development process



PART – C (2x15=30 Marks)

Answer all the questions

Q.No. **Questions**

7. a) Distinguish Software measurement in detail

- ✓ Classifying the entities to be examined
- ✓ Determining relevant measurement goals
- ✓ Identifying the level of maturity that the organization has reached
- ✓ Classifying the Entities to be Examined

In software engineering, mainly three classes of entities exist. They are –
Processes
Products
Resources
All of these entities have internal as well as external entities.

Internal attributes are those that can be measured purely in terms of the process, product, or resources itself. For example: Size, complexity, dependency among modules.

External attributes are those that can be measured only with respect to its relation with the environment. For example: The total number of failures experienced by a user, the length of time it takes to search the database and retrieve information.

The different attributes that can be measured for each of the entities are as follows –

Processes
Processes are collections of software-related activities. Following are some of the internal attributes that can be measured directly for a process –

The duration of the process or one of its activities

The effort associated with the process or one of its activities

The number of incidents of a specified type arising during the process or one of its activities

The different external attributes of a process are cost, controllability, effectiveness, quality and stability.

Products
Products are not only the items that the management is committed to deliver but also any artifact or document produced during the software life cycle.

The different internal product attributes are size, effort, cost, specification, length, functionality, modularity, reuse, redundancy, and syntactic correctness. Among these size, effort, and cost are relatively easy to measure than the others.

The different external product attributes are usability, integrity, efficiency, testability, reusability, portability, and interoperability. These attributes describe not only the code but also the other documents that support the development effort.

Resources
These are entities required by a process activity. It can be any input for the software production. It includes personnel, materials, tools and methods.

The different internal attributes for the resources are age, price, size, speed, memory size, temperature, etc. The different external attributes are productivity, experience, quality, usability, reliability, comfort etc.

Determining Relevant Measurement Goals
A particular measurement will be useful only if it helps to understand the process or one of its resultant products. The improvement in the process or products can be performed only when the project has clearly defined goals for processes and products. A clear understanding of goals can be used to generate suggested metrics for a given project in the context of a process maturity framework.

The Goal–Question–Metric (GQM) paradigm
The GQM approach provides a framework involving the following three steps –

Listing the major goals of the development or maintenance project

Deriving the questions from each goal that must be answered to determine if the goals are being met

Decide what must be measured in order to be able to answer the questions adequately

To use GQM paradigm, first we express the overall goals of the organization. Then, we generate the questions such that the answers are known so that we can determine whether the goals are being met. Later, analyze each question in terms of what measurement we need in order to answer each question.

Typical goals are expressed in terms of productivity, quality, risk, customer satisfaction, etc. Goals and questions are to be constructed in terms of their audience.

To help generate the goals, questions, and metrics, Basili & Rombach provided a series of templates.

Purpose – To (characterize, evaluate, predict, motivate, etc.) the (process, product, model, metric, etc.) in order to understand, assess, manage, engineer, learn, improve, etc. Example: To characterize the product in order to learn it.

Perspective – Examine the (cost, effectiveness, correctness, defects, changes, product measures, etc.) from the viewpoint of the developer, manager, customer, etc. Example: Examine the defects from the viewpoint of the customer.

Environment – The environment consists of the following: process factors, people factors, problem factors, methods, tools, constraints, etc. Example: The customers of this software are those who have no knowledge about the tools.

OR

b) Construct the process of Empirical model

Empirical modelling is a generic term for activities that create models by observation and experiment. Empirical Modelling (with the initial letters capitalised, and often abbreviated to EM) refers to a specific variety of empirical modelling in which models are constructed following particular principles. Though the extent to which these principles can be applied to model-building without computers is an interesting issue (to be revisited below), there are at least two good reasons to consider Empirical Modelling in the first instance as computer-based. Without doubt, computer technologies have had a transformative impact where the full exploitation of Empirical Modelling principles is concerned. What is more, the conception of Empirical Modelling has been closely associated with thinking about the role of the computer in model-building.

An empirical model operates on a simple semantic principle: the maker observes a close correspondence between the behaviour of the model and that of its referent. The crafting of this correspondence can be 'empirical' in a wide variety of senses: it may entail a trial-and-error process, may be based on computational approximation to analytic formulae, it may be derived as a black-box relation that affords no insight into 'why it works'.

Empirical Modelling is rooted on the key principle of William James's radical empiricism, which postulates that all knowing is rooted in connections that are given-in-experience. Empirical Modelling aspires to craft the correspondence between the model and its referent in such a way that its derivation can be traced to connections given-in-experience. Making connections in experience is an essentially individual human activity that requires skill and is highly context-dependent. Examples of such connections include: identifying familiar objects in the stream of thought, associating natural languages words with objects to which they refer, and subliminally interpreting the rows and columns of a spreadsheet as exam results of particular students in particular subjects.

8. a) Explain functional and nonfunctional requirements in software engineering

Non-Functional Requirements are the constraints or the requirements imposed on the system. They specify the quality attribute of the software. Non-Functional Requirements deal with issues like scalability, maintainability, performance, portability, security, reliability, and many more. Non-Functional Requirements address vital issues of quality for software systems. If NFRs not addressed properly, the results can include:

- Users, clients, and developers are unsatisfied.
- Inconsistent software.
- Time and cost overrun to fix the software which was prepared without keeping NFRs in mind.

Types of Non-functional Requirement :

- ✓ Scalability
- ✓ Reliability
- ✓ Regulatory
- ✓ Maintainability
- ✓ Serviceability
- ✓ Utility
- ✓ Security
- ✓ Manageability
- ✓ Data integrity
- ✓ Capacity
- ✓ Regulatory
- ✓ Availability
- ✓ Usability

- ✓ Interoperability
- ✓ Environmental

These can can be classified as :

1. **Performance constraints –**
   Reliability, security, response time, etc.
2. **Operating constraints –**
   These include physical constraints (size, weight), personnel availability, skill level considerations, system accessibility for maintenance, etc.
3. **Interface constraints –**
   These describe how the system is to interface with its environment, users, and other systems. For example, user interfaces and their qualities (e.g., user-friendliness).
4. **Economic constraints –**
   Immediate and/or long-term costs.
5. **Lifecycle requirements** – Quality of the design:
   These measured in terms such as maintainability, enhance ability, portability.

Advantages of Non-Functional Requirement :

- They ensure the software system follows legal and adherence rules.
- They specify the quality attribute of the software.
- They ensure the reliability, availability, performance, and scalability of the software system
- They help in constructing the security policy of the software system.
- They ensure good user experience, ease of operating the software, and minimize the cost factor.

Disadvantages of Non-functional requirement :

- The nonfunctional requirement may affect the various high-level software subsystem.
- They generally increase the cost as they require special consideration during the software architecture/high-level design phase.
- It is difficult to change or alter non-functional requirements once you pass them to the architecture phase.

OR

b) Discuss the followings

black box testing

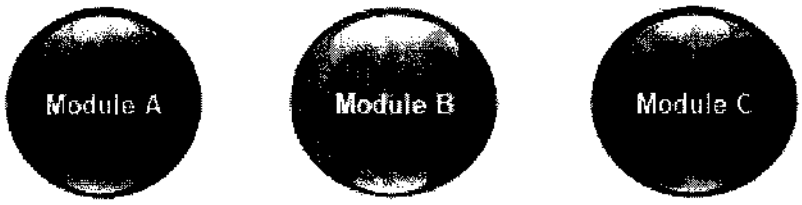
Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing.

Initially, the requirements and specifications of the system are examined.
Tester chooses valid inputs (positive test scenario) to check whether SUT processes them correctly. Also, some invalid inputs (negative test scenario) are chosen to verify that the SUT is able to detect them.
Tester determines expected outputs for all those inputs.
Software tester constructs test cases with the selected inputs.

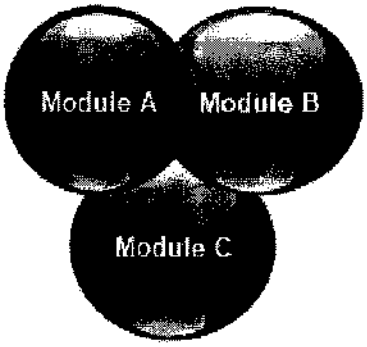
The test cases are executed.
Software tester compares the actual outputs with the expected outputs.
Defects if any are fixed and re-tested.

Integration Testing

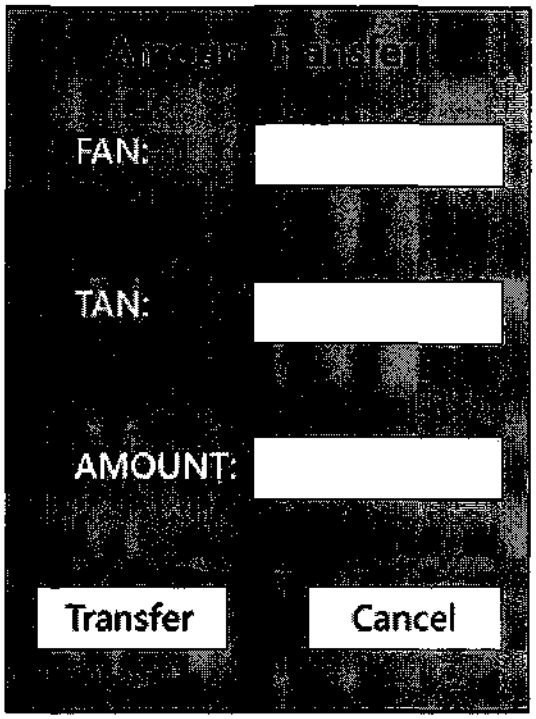
Integration testing is the phase in software testing in which individual software modules are combined and tested as a group. Integration testing is conducted to evaluate the compliance of a system or component with specified functional requirements. It occurs after unit testing and before validation testing

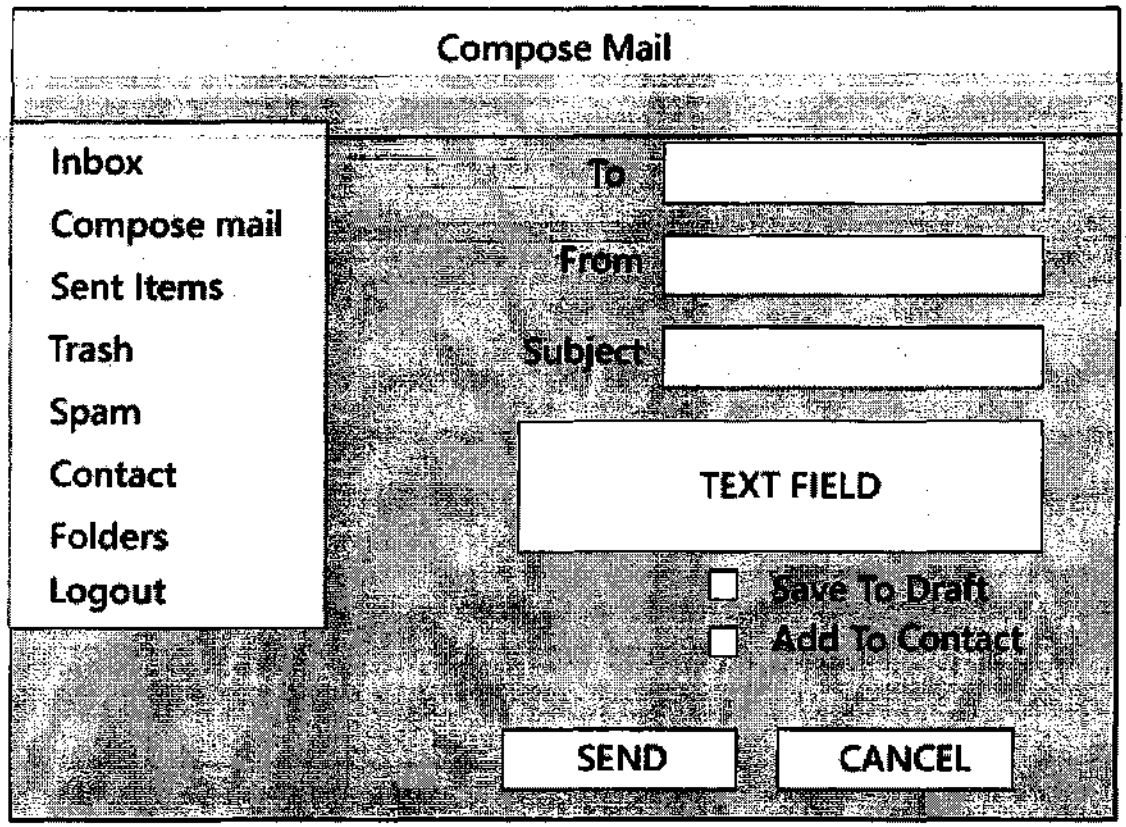


Tested in Unit Testing



Under Integration Testing





Scenarios1:

- First, we login as **P** users and click on the **Compose** mail and performing the functional testing for the specific components.
- Now we click on the **Send** and also check for **Save Drafts**.
- After that, we send a **mail** to **Q** and verify in the **Send Items** folder of P to check if the send mail is there.
- Now, we will **log out** as P and login as Q and move to the **Inbox** and verify that if the mail has reached.

Black Box Testing

- State Transition technique
- Decision Table Technique
- Boundary Value Analysis
- All-pairs Testing
- Cause and Effect Graph
- Equivalence Partitioning
- Error Guessing

White Box Testing

- Data flow testing
- Control Flow Testing
- Branch Coverage Testing
- Decision Coverage Testing

21/5/2021
Course Faculty

HOD/CSE

CSE-B

Name : S. Sarandhar
Roll No : 19CS113
Year : 2nd/IV B
Sub : Software Engineering
Date : 20th may 2021

Part A

1. Steps involved in Software Designing process

1) Understanding project requirements
2) Research and Analysis
3) Design
4) Prototyping
5) Evaluation

2. Decomposability

When software can be decomposed into independent modules, these modules can be tested individually. When an error occurs in an individual module, the error is less likely to require changes to be made in other modules or for the tester to even examine multiple modules.

1) Regression Testing

Regression testing is the process of testing the modified parts of the code and the parts that might get affected due to the modification. They ensure that no new errors have been introduced in the software after the modification have been made.

2) Refactoring

Refactoring simply means to reconstruct something in such a way that it does not affect the behaviour or any other features

3)

Testing	Validation
It is an investigation conducted to provide stakeholders with information about quality of product	It is process of checking whether the software product is up to the mark in terms of user need and requirements.

Part-3

6) Architectural design in software development

⇒ The software architecture is the overall structure of the system and it includes the software components, their properties, and the relationships among the components

⇒ Software architectural design represents the structure of the data and program components that are required to build a computer based system

⇒ An architectural design model is transferable

* It can be applied to the design of other system

* It represents a set of abstractions that enable software engineers to describe architecture in predictable ways

⇒ For describing the architecture, there should be need of two processing to derive the architecture

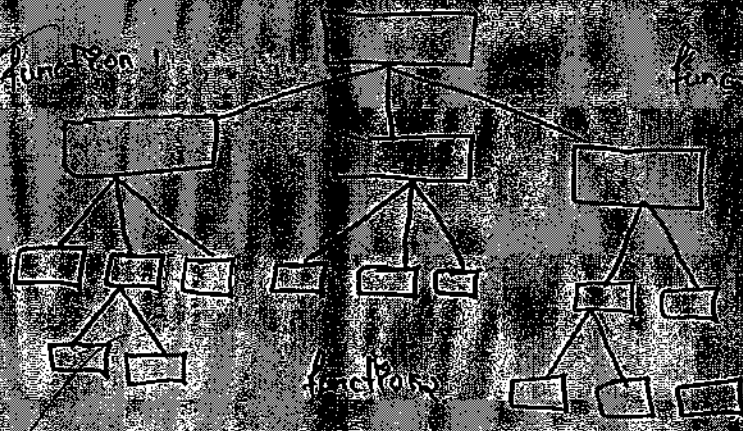
1) Horizontal partitioning

2) Vertical partitioning

1) Horizontal partitioning

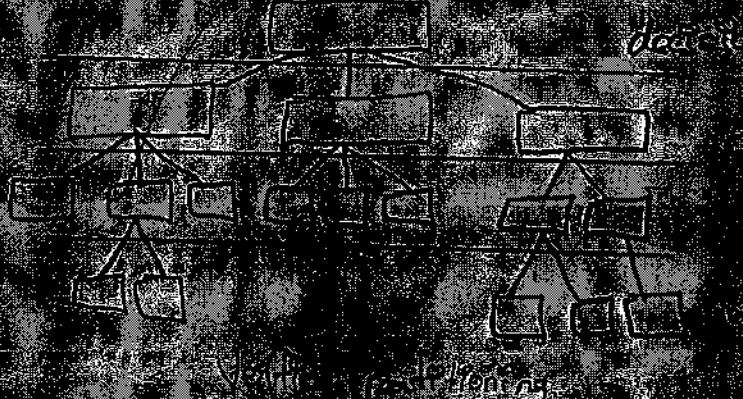
* Horizontal partitioning define separate branches of the module hierarchy for each of the major functions.
* This use control module to coordinate communication between functions

eg.



2) Vertical partitioning

* Vertical partitioning suggest the control and work should be distributed top down in the program structure
* The decision making module should reside at the top of the architecture



Part-C

9a) Software Measurement

A measurement is an manifestation of the size, quality, amount or dimension of a particular attributes of a product or process. Software measurement is a private impute of a characteristic of a software product or software process. It is an authority within software engineering. Software measurement process is defined and governed by ISO standard.

Need of Software Measurement

1) Create the quality of the current product or process.
2) Anticipate future qualities of the product or process.
3) Enhance the quality of product or process.
4) Regulate the state of the project relative to budget and schedule.

Classification of software Measurement

There are two types of software measurement are they are,

1) Direct Measurement

2) Indirect Measurement

1) Direct Measurement

In direct measurement the product, process or thing is measured directly using standard scale

2) Indirect Measurement

In indirect measurement the quantity or quality to be measured is measured using related parameter i.e. use of reference.

Metrics

A metrics is a measurement of the level that any input belongs to a system product process that are four functional related to software metrics

1) Planning

2) Planning

3) Controlling

4) Improving

Characteristics of Software Metrics

1) Quantitative

Metrics must possess quantitative nature. It means metrics can be expressed in values.

2) Understandable

Metric computation should be easily understood, the method of computing metric should be clearly defined.

3) Applicability

Metrics should be applicable in the initial phase of development of the software.

4) Repeatable

The metric values should be same when measured repeatedly and consistent in nature.

5) Economical

Computation of metrics should be economical.

6) Language Independent

Metrics

programming language

Classification of Metrics

1) Product Metrics

Product metrics are used to evaluate the state of the product, tracing risks and undercovering problem areas. The ability of team to control quality is evaluated

2) Process Metrics

Process metrics pay particular attention on enhancing the long term process of the team.

3) Project Metrics

It describes the project characteristic and execution process

* No. of software developers
* Staffing pattern over the life cycle of software
* Cost and schedule
* Productivity

8) b) Black Box testing.

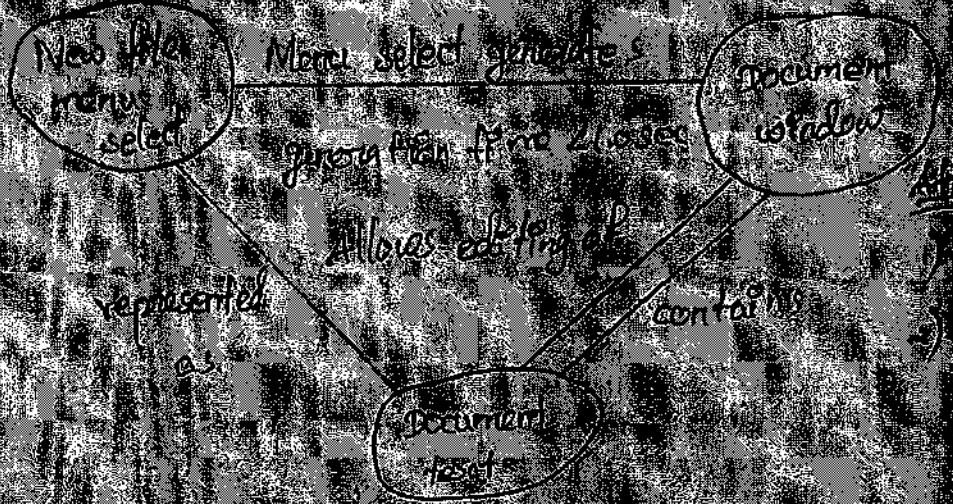
This testing focuses on the functional requirements of the software and the information domain of the software. It is also called as behavioral testing. Black box testing attempts to find errors on following categories

* Incorrect or missing functions
* Interface errors
* Errors in data structure
* Performance errors
* Initialisation and termination errors

A set of input conditions are identified to check all functional requirements.

Graph based testing Methods

* The first step in black box testing is to understand the objects and the relationship
* Then a series of tests are defined to verify whether all the objects have the expected relationship with each other



Behaviour testing methods can be made use of graphs.

1) Transaction flow modeling

The nodes represent steps in some transaction and the links represent the logical connection between steps.

2) Finite state modeling

The nodes represent different user observable states of the software and the link represent the transition that occur to move from the state to state.

3) Data flow modelling

the nodes are data objects and the links are the transformation that occur to translate one data object into another one.

4) Timing Modelling.

The nodes are program objects and the links are sequenced connection between those objects. Timing weights are used to specify the required execution times.

5) Integration Testing.

It is defined as systematic technique for constructing the software architecture and also for conducting tests to uncover errors associated with the interface. Two approaches get used

a) Big Bang Approach

b) Incremental integration testing

Big Bang Approach.

* In this approach all the components are combined in advance and the entire program is tested as a whole.

* If this is not appropriate and errors are encountered, correction is difficult because isolation of causes is complicated.

2) Incremental Integration Testing

* 3 types of these Approaches they are

1) Top down Integration

2) Bottom up Integration

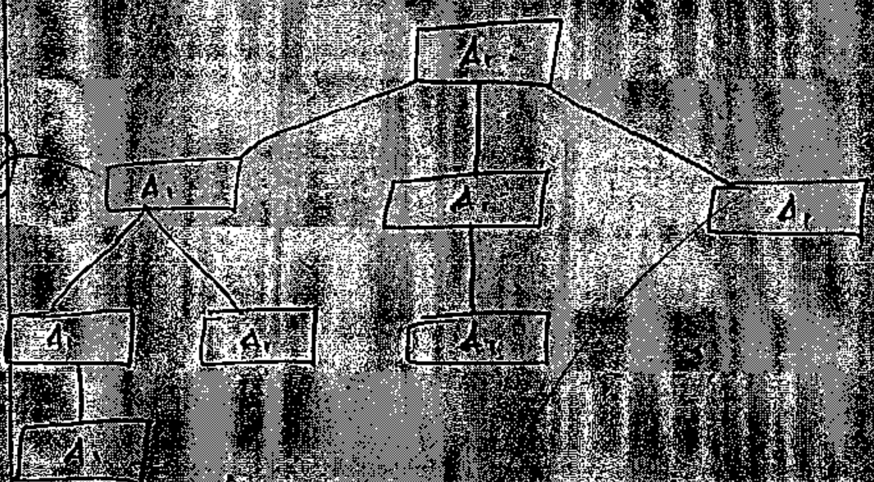
3) Sandwich Integration

* Errors are easier to isolate and correct.

* Interfaces are more likely to be tested completely.

* A systematic test approach is applied.

Steps followed in top down Integration



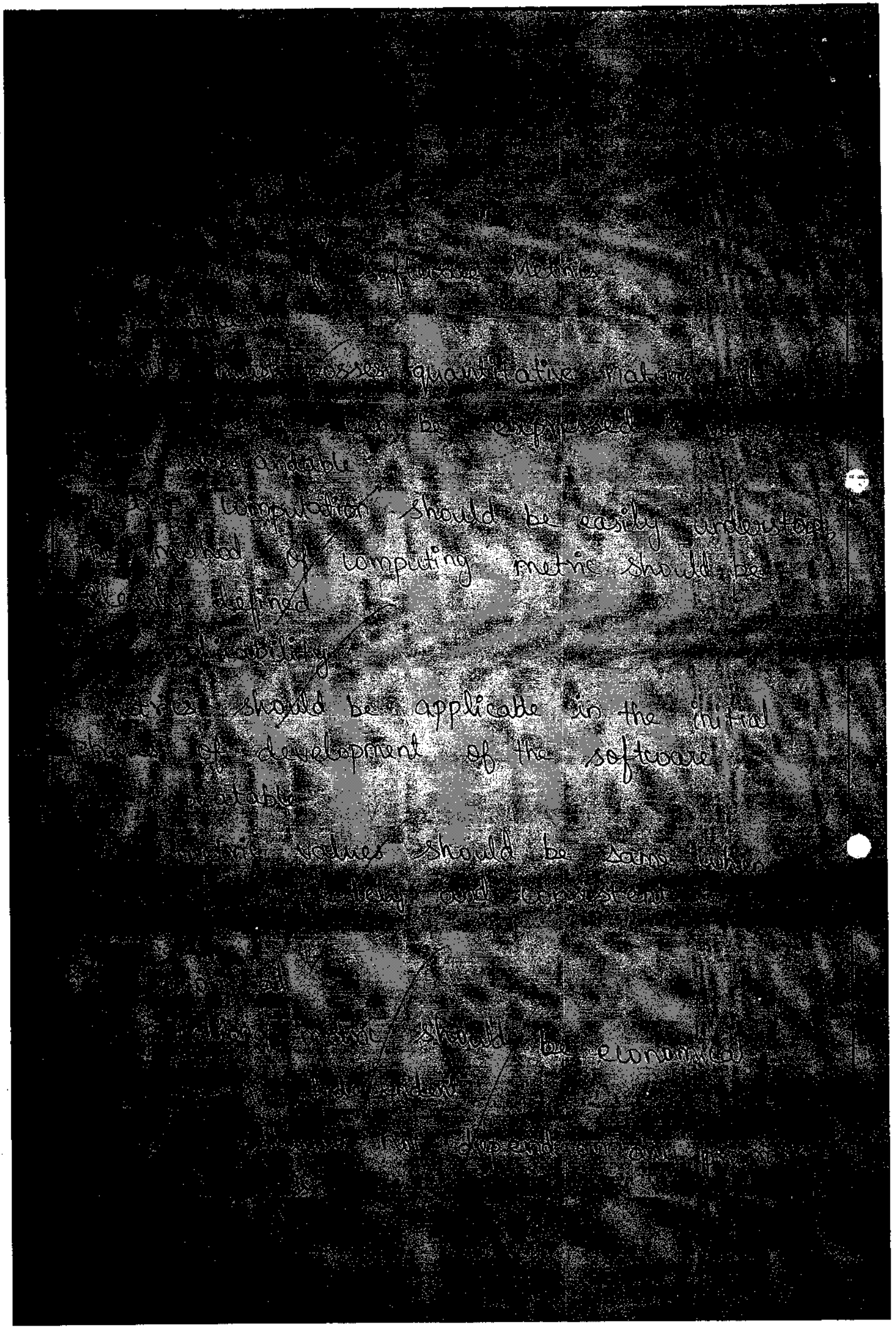
* The main control module is used as a test driver and stubs.

* Depending on the integration approach selected subordinate stubs

* Tests are conducted as each component is integrated.

* On completion of each set of tests another is replaced with real one.

* Regression testing may conduct to ensure the new errors have not been introduced.



... of Software Metric

... of the product, tracking ...
... prospective problem area ...
... of team to ...

... Process Metrics

... particular attention on enhancing ... long term process of the team or organis...

Project Metrics

... project characteristic and execution process

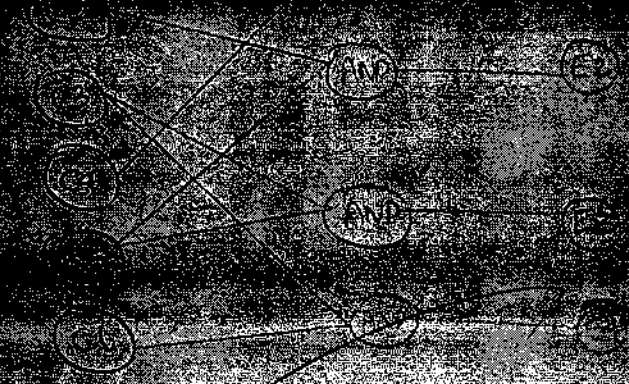
* No. of software developer
* Cost and schedule
* Productivity

Black box Testing

Black box testing is a type of software testing in which the functionality of the software is not known.

Black box testing can be done in ... ways

Syntax Driven Testing – This type ... testing is applied to systems that ...

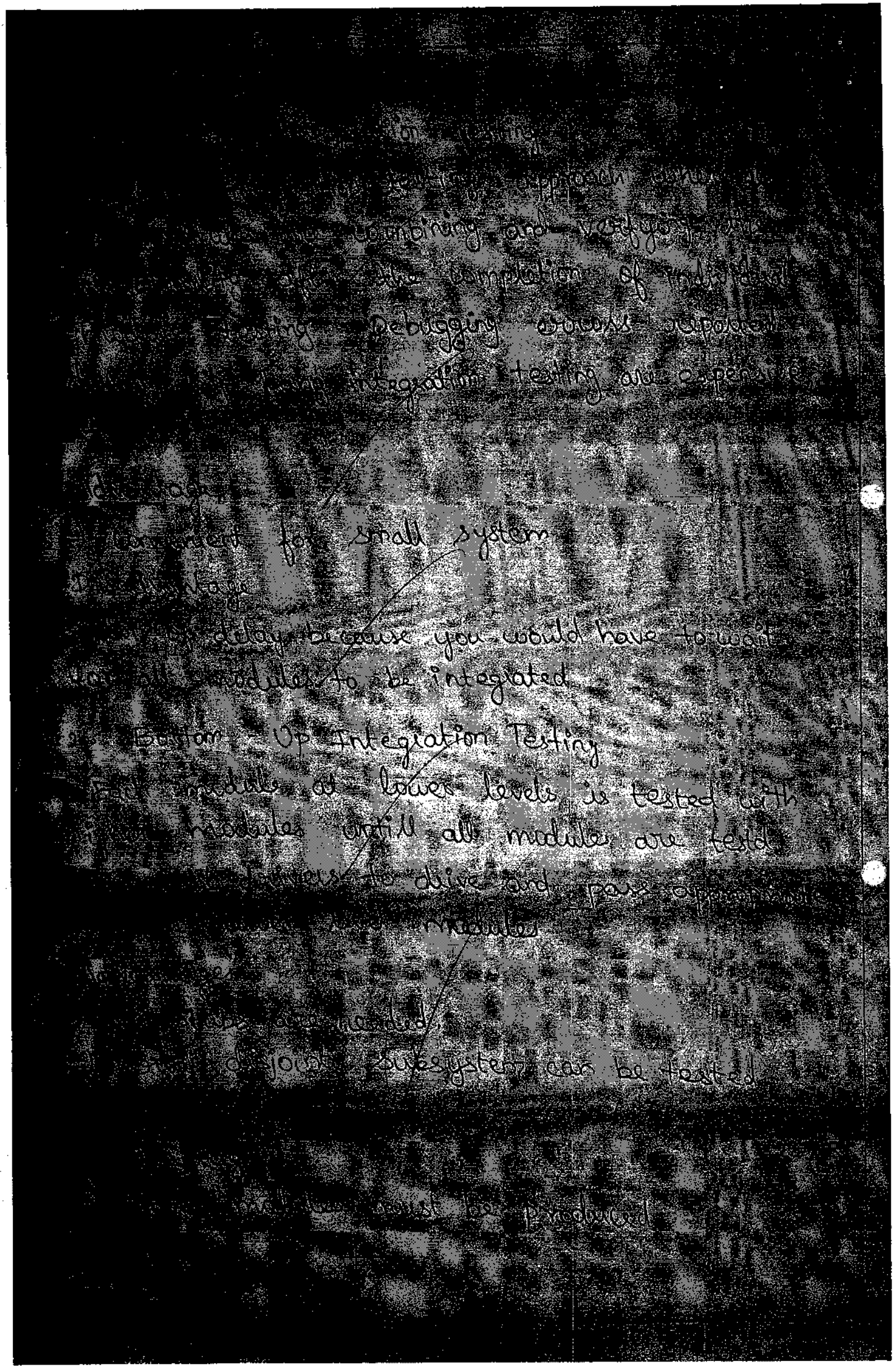


Can be converted into decision table

		1	2	3	4
Causes	C1	1	0	0	0
	C2	0	1	0	0
	C3	0	0	1	1
	C4	1	0	0	0
	C5	0	1	1	0
	C6	0	0	0	0
Effects	E1	X	–	–	–
	E2	–	X	–	–
	E3	–	–	X	–
	E4	–	–	–	X

Integration Testing

Q. The types of testing that occur between two software units or



Bottom Up Integration Testing

Advantages

Disadvantages

Sandwiched Integration Testing

Advantages

Disadvantages

Architectural Design in Software Engineering

Software Development Process

includes the software

properties and the relationship

components

architectural design represents

of the data and program components

required to build a computer based

Architectural Design model is transferable

can be applied to the design of other

represents a set of abstractions that

software engineers to describe architecture

predictable ways

for structuring the architecture, horizontal and

partitioning are required

partitioning

partitioning defines separate branches

hierarchy for each major

modules to coordinate

Vertical partitioning - Factoring

=> Vertical partitioning suggests the control [illegible] should be distributed [illegible] the program structure

=> The decision making modules [illegible] at the top of the Architecture

Fig: Vertical Partitioning

CSE-B

G. SARAVANAN

19CS109

SE - 19CSC12

20.05.2021

Online Exam

30

Part - A

1. The step involved in Software Designing Process:

Stage 1 : Understanding Project requirements

Stage 2 : Research and Analysis

Stage 3 : Design

Stage 4 : Prototyping

Stage 5 : Evaluation

2. State Decompasability

When software can be decomposed into independent modules, these modules can be tested individually. When an error occurs in an individual module the error is less likely to require change to be made in other module or for tested to even examine multiple module

3. Regression testing

The process of testing the modified parts of the code and the parts that might get affected due to the modifications to ensure that new error and affected bugs are not introduced in software product after the modifications made

5 Refactoring

Refactoring simply means to reconstruct something in such a way that it does not affect the behaviour or any other features

8 Testing is important for any software development life cycle it is fundamental to ensure of any quality and to have a work for software where as validation refers to focusing on staying the interface it should be able to perform tasks correctly and is should be handled a task

Part -B

6 Architectural Design in Software development process

Architectural Design

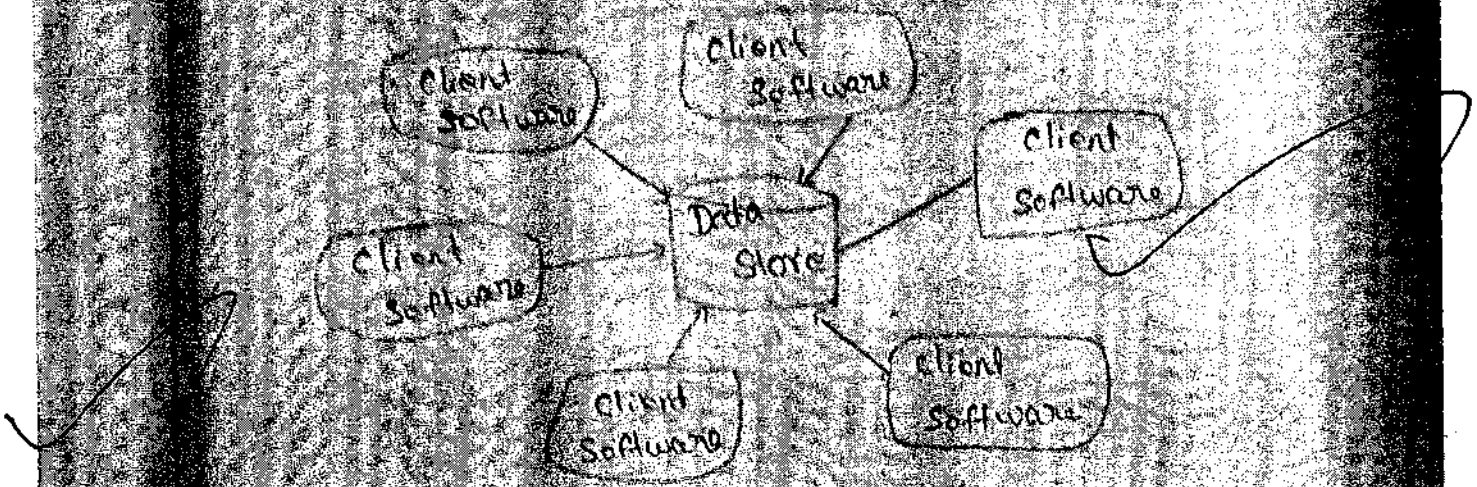
Introduction: The software needs the architectural design to represent the design of software. IEEE defines architectural design as the process of defining a collection of hardware and software components and their interfaces to establish framework for the development of a computer system. The software that is built for computer based system can exhibit one of many architectural styles.

Taxonomy of Architectural Styles:

1. Data centred architectures:

A data store will reside at the centre of this architecture and is accessed frequently by the other component that update, add, delete or modify the data present within the store.

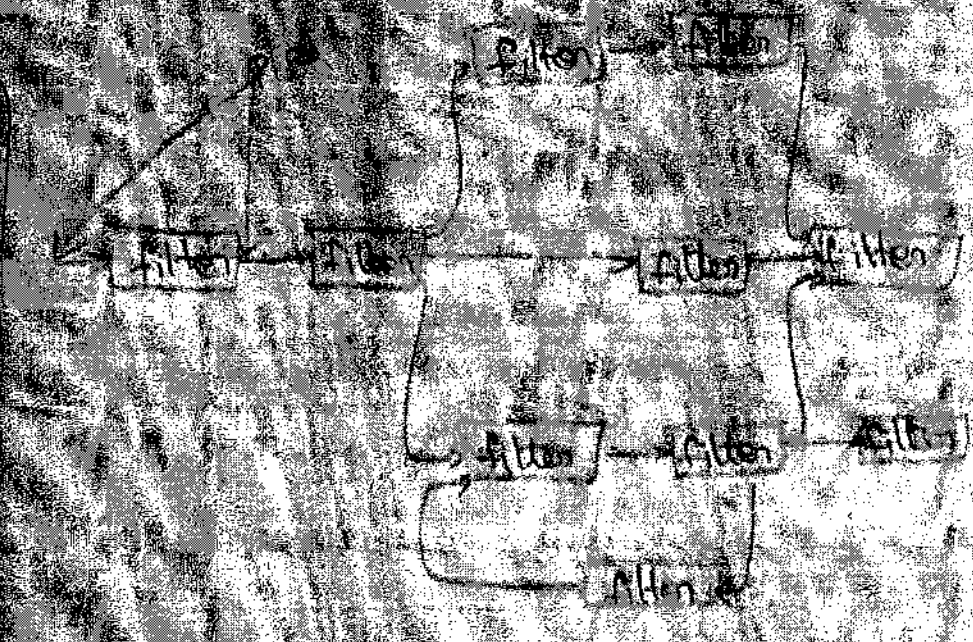
Data can be passed among client using blackboard mechanism.



2. Data flow architectures:

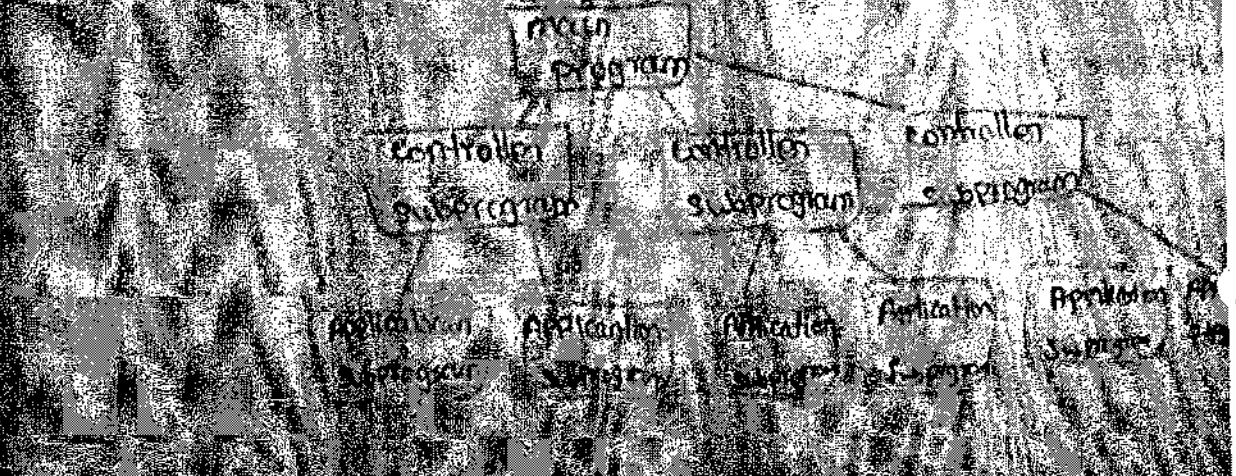
This kind of architecture is used when input data to be transformed into output data through a series of computational manipulative components.

Pipes are used to transmit data from one component to next component.



3. Call and Return architecture:-

* Remote procedure call architecture
* Main program or subprogram architecture



4. Object Oriented Architecture

5. Layered Architecture

Part - C

7. a) Software measurement :-

A measurement is an manifestation of the size quantity amount or dimension of a particular attributes of Product or process. software measurement is a titrate impute of a characteristic of a Software Product or the software process. It is an authority within software engineering. Software measurement process is defined and governed by ISO standards.

Need of Software measurement :-

1. Create the quality of the current product or process.
2. Anticipate feture qualities of the product or process.
3. Enhance the quality of a product or process.
4. Regulate the state of the project in relation to budget and schedule.

Classification of Software measurement :-

1. Direct Measurement
2. Indirect Measurement

Metrics :-

Metrics is a measurement of the level that any impute belongs to a system product or process. There are different types of metrics which are listed below :-

1. Planning
2. Organizing
3. Controlling
4. Improving

Characteristics of software Metrics

1. Quantitative
Metrics must possess quantitative nature it
mean metrics can be expressed in values.

2. Understandable
Computation should be easily understood,
the method of computing metric should be
clearly defined

3. Applicability
Applicable in the initial phases of develop
of the software

4. Repeatable
Same when measured repeatedly and consis
in nature

5. Economical
Computation of Metrics should be economical

6. Language independent
Not depend on any programming language

Classification of Software Metrics:-

(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC, NBA & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu.
Estd. 2000

OFFICE OF THE CONTROLLER OF EXAMINATIONS

CIA-III-SET-B

Degree & Branch	:	**B.E & CSE**	**Max Marks**	:	**50**
Year/Sem	:	**II/IV**	**Duration**	:	**3.00 Hrs**
Course Code & Course Name	:	**19CSC12/SOFTWARE ENGINEERING**	**Date**	:	**10.05.2021**

19CSC12.CO1	Apply the concepts of life cycle models to choose the appropriate model
19CSC12.CO2	Analysis the requirements and design the software
19CSC12.CO3	Construct a design for a real-world problem
19CSC12.CO4	Design and develop test cases
19CSC12.CO5	Work with version control and work on configuration and release management plans

PART-A (5x2=10Marks)
Answer all the questions

Q.No.	Questions	BT Level	Course Outcome
1.	Define software engineering?	K1	CO1
2.	List out the available software risk.	K1	CO1
3.	Discover the levels of phases of design.	K1	CO2
4.	Distinguish between verification and validation.	K2	CO2
5.	Rewrite black box testing	K2	CO3
6.	State RFP	K1	CO3
7.	Define control structure testing	K1	CO4
8.	How to compute the Cyclomatic complexity?	K3	CO4
9.	Test COCOMO II Model.	K4	CO5
10.	Synthesize the factors that lead to Risk?	K4	CO5

PART – C (5x16=80 Marks)
Answer all the questions

Q.No.		Questions	BT Level	Course Outcome
11.	a)	Describe the golden rules for interface design	K1	CO1
		OR		
	b)	Construct the process of Empirical model	K5	CO1
12.	a)	Explain functional and nonfunctional requirements in software engineering	K2	CO2
		OR		
	b)	Illustrate the basic structure of analysis model	K3	CO2

necessary block diagrams

OR

	b)	Reconstruct the Taxonomy of Architectural styles	K5	CO3
14.	a)	ReWrite short notes on a) Data flow testing. b) Integration testing	K5	CO4
		OR		
	b)	Discuss the differences between black box and white box testing models. Discuss how these testing models may be used together to test a program schedule.	K2	CO4
15.	a)	Analyze the needs and activities or risk management.	K4	CO5
		OR		
	b)	Discuss on the various software cost estimation techniques	K2	CO5

Bloom's Taxonomy Level

Bloom's Taxonomy Level	K1 Remembering	K2 Understanding	K3 Applying	K4 Analyzing	K5 Evaluating	K6 Creating	Total
% of Questions	14	38	10	11	27	0	100

1. 9/5/21
2. 9/5/21

Course Faculty **Course Coordinator** **HoD**

9/5/21 9/5/21

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC, NBA & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu.
Estd. 2000

OFFICE OF THE CONTROLLER OF EXAMINATIONS

CIA-III-SET-A

ANSWER KEY

Degree & Branch	:	**B.E & CSE**	**Max Marks**	:	**100**
Year/Sem	:	**II/IV**	**Duration**	:	**3.00 Hrs**
Course Code & Course Name	:	**19CSC12/SOFTWARE ENGINEERING**	**Date**	:	**10.05.2021**

16CSD12.CO1	:	Apply the concepts of life cycle models to choose the appropriate model
16CSD12.CO2	:	Analysis the requirements and design the software
16CSD12.CO3	:	Construct a design for a real-world problem
16CSD12.CO4	:	Design and develop test cases
16CSD12.CO5	:	Work with version control and work on configuration and release management plans

PART-A (5x2=10Marks)
Answer all the questions

Q.No **Questions**

1. **Define software engineering?**

   Systematic application of engineering approaches to the development of software.

2. **List out the available software risk.**

   Internal risks that are within the control of the project manager and

   External risks that are beyond the control of project manager

3. **Mention the levels of phases of design.**

   - ✓ Interface Design
   - ✓ Architectural Design
   - ✓ Detailed Design

4. **Distinguish between verification and validation.**

Verification	Validation
• The verifying process includes checking documents, design, code, and program	• It is a dynamic mechanism of testing • validating the actual product

5. **Rewrite black box testing**

   - Demonstrate that software functions are operational
   - That input is properly accepted and output is correctly produced
   - Integrity of external information

- Request for proposal
- Part of the bidding procedure for a product or service.
- To provide a structured way for companies

7. **Define control structure testing**

Control structure testing is a group of white-box testing methods. CONDITION TESTING. It is a test case design method. It works on logical conditions in program module. It involves testing of both relational expressions and arithmetic expressions.

8. **How to compute the cyclomatic complexity?**

The MSDN states: "Cyclomatic complexity measures the number of linearly independent paths through the method, which is determined by the number and complexity of conditional branches.

Here's how cyclomatic complexity is calculated:

CC = E - N + 1.

9. **Reproduce COCOMO II Model.**

COCOMO-II is the revised version of the original Cocomo (Constructive Cost Model) and is developed at University of Southern California. It is the model that allows one to estimate the cost, effort and schedule when planning a new software development activity.

10. **List the factors that lead to Risk?**

The size of the sale.

The number of people who will be affected by the buying decision.

The length of life of the product.

The customer's unfamiliarity with you, your company, and your product or service.

11. a) **Explain in detail about the software process Model.**

A software process model is a simplified representation of a software process. Each model represents a process from a specific perspective.(4 Marks)

Waterfall Model.

- The waterfall model is a sequential approach, where each fundamental activity of a process represented as a separate phase, arranged in linear order.

Prototyping Model.

- A prototype is useful when a customer or developer is not sure of the requirements, or of algorithms, efficiency, business rules, response time, etc.

Spiral Model.

The spiral model is a risk-driven where the process is represented as spiral rather than a sequence of activities.

OR

b) **Summarize software project management.**

Software Project Management (SPM) is a proper way of planning and leading software projects. It is a part of project management in which software projects are planned, implemented, monitored and controlled.

Need of Software Project Management:

Software is an non-physical product. Software development is a new stream in business and there is very little experience in building software products. Most of the software products are made to fit client's requirements. The most important is that the basic technology changes and advances so frequently and rapidly that experience of one product may not be applied to the other one. Such type of business and environmental constraints increase risk in software development hence it is essential to manage software projects efficiently.

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

Requirement Engineering Process

It is a five step process, which includes –

Feasibility Study

Requirement Elicitation and Analysis

- Software Requirement Specification
- Software Requirement Validation
- Software Requirement Management

OR

b. Illustrate the basic structure of analysis model.

Elements of the analysis model

1. Scenario based element

This type of element represents the system user point of view.

Scenario based elements are use case diagram, user stories.

2. Class based elements

The object of this type of element manipulated by the system.It defines the object,attrib relationship.The collaboration is occurring between the classes.Class based elements ar class diagram, collaboration diagram.

3. Behavioral elements

Behavioral elements represent state of the system and how it is changed by the external The behavioral elements are sequenced diagram, state diagram.

4. Flow oriented elements

An information flows through a computer-based system it gets transformed.

It shows how the data objects are transformed while they flow between the various syst functions. The flow elements are data flow diagram, control flow diagram.

13. a. Explain the core activities involved in User Interface design process with necessary block diagrams

User interface is the front-end application view to which user interacts in order to use the software. .(4 Marks)

Types of User Interface

1. **Command Line Interface.**

OR

b. Reconstruct the Taxonomy of Architectural styles

- Data centered architectures(4 Marks)
- Data flow architectures(3 Marks)
- Call and Return architectures(3 Marks)
- Object Oriented architecture(3 Marks)
- Layered architecture(3 Marks)

14.a. **ReWrite short notes on**

a) Data flow testing.

b) Integration testing

Data flow testing(8 Marks)

This testing technique emphasis to cover all the data variables included in the program. It tests where the variables were declared and defined and where they were used or changed.

Integration testing(8 Marks)

Integration testing is the process of testing the interface between two software units or module. It's focus on determining the correctness of the interface. The purpose of the integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

OR

b) **Discuss the differences between black box and white box testing models. how these testing models may be used together to test a program schedul**

Black Box Testing is a software testing method in which the internal structur implementation of the item being tested is not known to the tester

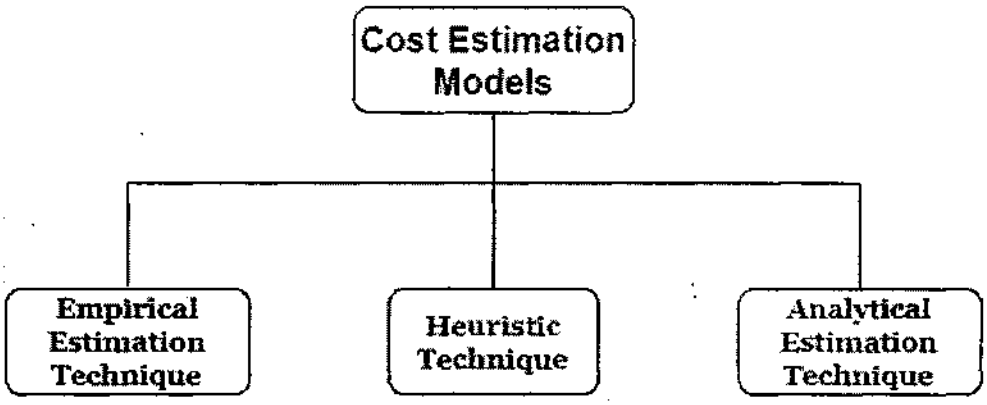
White Box Testing is a software testing method in which the internal structu implementation of the item being tested is known to the tester.

It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
It is mostly done by software testers.	It is mostly done by software developers.
No knowledge of implementation is needed.	Knowledge of implementation is required.
It can be referred as outer or external software testing.	It is the inner or the internal software testing.
It is functional test of the software.	It is structural test of the software.
This testing can be initiated on the basis of requirement specifications document.	This type of testing of software is started after detail design document.
No knowledge of programming is required.	It is mandatory to have knowledge of programming.
It is the behavior testing of the software.	It is the logic testing of the software.
It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
It is also called closed testing.	It is also called as clear box testing.
It is least time consuming.	It is most time consuming.
It is not suitable or preferred for algorithm testing.	It is suitable for algorithm testing.
Can be done by trial and error ways and methods.	Data domains along with inner or internal boundaries can be better tested.
Example: search something on google by using keywords	**Example:** by input to check and verify loops

Project Risks :
These are the risks which affect the project schedule or resources.
Product Risks :
These are the risks which affect the quality or performance of the being developed.
Business Risks :
These are the risks which affect the organization developing or procuring the software.

OR

b) **Discuss on the various software cost estimation techniques**



Course Faculty

HoD

19CSC15

CIA-3 SE 10/5/21

PART-A.

95/100

1. Software Engineering:

Systematic application of engineering approaches to the development of software.

2. the available software risk:

Internal risks that are within the control of the project manager and external risks that are beyond the control of project manager.

3. the levels of phases of design:

* Interface Design
* Architectural Design
* Detailed Design

4.

Verification	Validation.
*. the verifying process includes checking documents, design, code and program.	*. It is a dynamic mechanism of testing *. validating the actual product.

5. Black box testing:

*. Demonstrate that software functions are operational.
*. that input is properly accepted and output is correctly produced.
*. Integrity of external information.

6. RFP:

*. Request for proposal
*. Part of the bidding procedure for a product of savice.
*. to provide a structured way for companies.

7. control structure testing:

⇒. control structure testing is a group of white-box testing methods
⇒. CONDITION TESTING
⇒. It is a test case design method.
⇒. It works on logical conditions in program module.

8. the cyclomatic complexity:

the MSDN states, "cyclomatic complexity measures the number of linearly independent paths through the method, which is determined by the number and complexity of conditional branches. $CC = E - N + 1$.

9. COCOMO II Model:

⇒. COCOMO-II is the revised version of the original Cocomo and is developed at University of Southern California.

⇒. It is the model that allows one to estimate the cost, effort and schedule when planning a new software development activity.

10. the factors that lead to risk:

*. the size of the scale.

*. the number of people who will be affected by the buying decision.

*. the length of life of the product.

*. the customer's unfamiliarity with you, your company, and your product or service.

Part-B.

11.

a). the software process model:

⇒ A software process model is an abstract representation of the development process.

⇒. A models specify the stages and order of a process.

⇒. so, think of this as a representation of the order of activities of the process and the sequence in which they are performed.

⇒. the tasks to be performed.

⇒. the input and output of each task.

⇒. the pre and post conditions for each task.

⇒. the flow and sequence of each task.

⇒. the goal of a software process model is to provide guidance for controlling and coordinating the tasks to achieve the end product and objectives as effectively as possible.

⇒. there are many kinds of process models for meeting different requirements.

⇒. we refer to these as SDLC models.

⇒. the most popular and important SDLC models are as follows:

*. waterfall model
*. V model
*. Incremental model
*. RAD model
*. Agile model
*. Iterative model
*. Prototype model
*. Spiral model

⇒. choosing the right software process model for your projects can be difficult.

⇒. If you know your requirements well, it will be easier to select a model that best matches your needs.

⇒. You need to keep the following factors in mind when selecting your software process model:

Project size: consider the size of the project you will be working on. Larger projects mean bigger teams, so you'll need more extensive and elaborate project management plans.

Project complexity: complex projects may not have clear requirements. the requirements may change often, and the cost of delay is high.

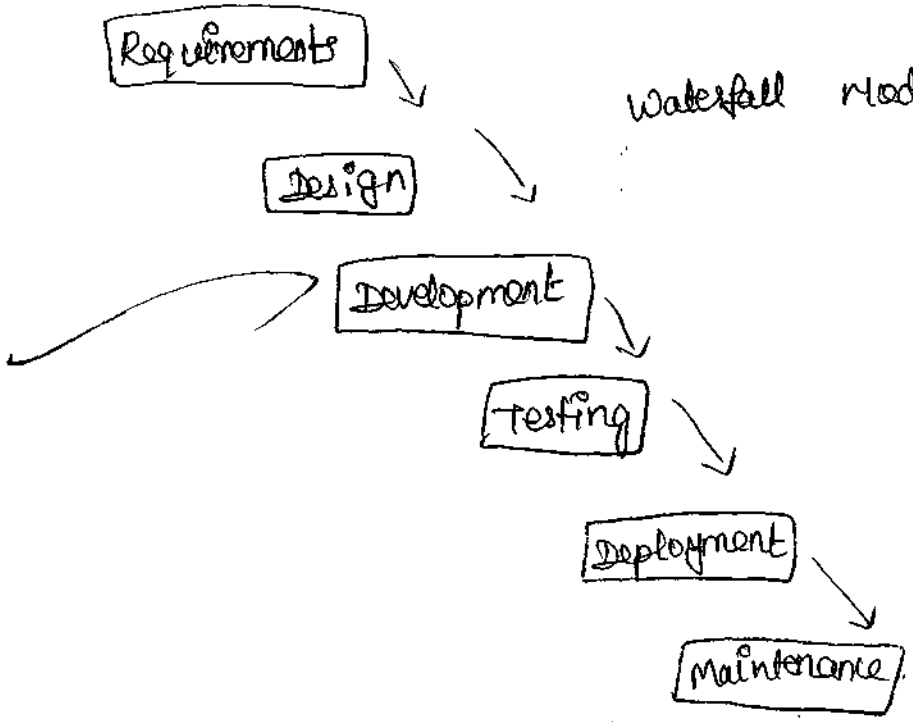
cost of delay: Is the project highly time-bound with a huge cost of delay.

Familiarity with technology: this involves the developers' knowledge and experience with the project domain, software tools, language and methods needed for development.

Project resources: this involves the amount and availability of funds, staffs and other resources.

⇒. The waterfall model is a breakdown of project activities into linear sequential phases, where each phase depends on the deliverables of the previous one and corresponds to a specialisation of tasks.

⇒. The approach is typical for certain areas of engineering design.



Waterfall model.

⇒. the V-model represents a development process that may be considered an extension of the waterfall model and is an example of the more general V-model.

⇒. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase to form the typical V shape.

⇒. the V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing.

⇒. the horizontal and vertical axes represent time or project completeness and level of abstraction respectively.

12.

a). Requirement engineering process:

⇒. Requirement Engineering is the process of defining, documenting and maintaining the requirements.

⇒. It is like a process of gathering and defining service provided by the system.

⇒. Requirements Engineering Process consists of the following main activities:

* Requirements elicitation
* Requirements specification
* Requirements verification and validation
* Requirements management.

Requirements elicitation:

⇒. It is related to the various ways used to gain knowledge about the project domain and requirements.

⇒. the various source of domain knowledge include customers, business manuals, the existing software of same type, standards and other stakeholders of the project.

Requirements specification:

⇒. this activity is used to produce formal software requirement models.

⇒. All the requirements including the functional as well as the non-functional requirements and the constraints are specified by these models in totality.

⇒. During specification, more knowledge about the problem may be required which can again trigger the elicitation process.

⇒. the models used at this stage include ER diagrams, data flow diagrams.

Requirements verification and validation:

⇒. It refers to the set of tasks that ensures that the software correctly implements a specific function.

⇒. It refers to a different set of tasks that ensures that the software that has been built is traceable to customer requirements.

⇒. If requirements are not validated, errors in the requirement definitions could propagate to the succesive stages resulting in a lot of modification and rework.

⇒. The main steps for this process include:

*. the requirements should be consistent with all the other requirements.

*. the requirements should be complete in every other.

*. the requirements should be practically achievable.

Reviews, buddy checks, making test cases etc.. are some of the methods used for this.

Requirements management:

*. Requirement management is the process of analyzing, documenting, tracking, prioritizing and agreeing the requirement and controlling the communication to relevant stakeholders.

*. Being able to modify the software as per requirements in a systematic and controlled manner is an extremely important part of the requirements engineering process.

13

b) The taxonomy of Architectural styles:

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

⇒. Each style describes a system category that encompasses

*. (1) A set of components that perform a function required by a system;

*. (2) A set of connectors that enable "communication, coordination and cooperation" among components;

*. (3) Constraints that define how components can be integrated to form the system;

*. (4) Semantic models that enable a designer to understand the overall properties of a system

by analyzing the known properties of its constituent parts.

⇒ The following are the taxonomy of Architectural style

* Data-centered architectures
* Data flow architectures
* Call and return architectures
* object-oriented architectures
* Layered architectures.

Data-centered Architecture:

⇒ A data store 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.

⇒ Data-centered architectures promote inerrability. that is, existing components can be changed and new client components added to the architecture without concern about other clients.

⇒ client components independently execute processes.

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.

⇒ However, the filter does not require knowledge of the working of its neighbouring filters.

Call and return architectures:

⇒ This architectural style enables you to achieve a program structure that is relatively easy to modify and scale.

⇒. A number of substyles exist within this category.

⇒. Main program / subprogram architectures.

⇒. Remote procedure call architectures.

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 are accomplished via message passing...

Layered architectures:

⇒. The basic structure of a layered architecture is Illustrated

⇒. At the outer layer, components service user interface operations.

⇒. At the inner layer, components perform operating system interfacing.

14.

b).

Black Box testing	White Box Testing
*. It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	*. It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
*. It is mostly done by software testers.	*. It is mostly done by software developers.
*. No knowledge of implementation is needed.	*. Knowledge of implementation is required.
*. It can be referred as outer or external software testing.	*. It is the inner of the internal software testing.

*. It is functional test of the software.	*. It is structural test of the software.
*. this testing can be initiated on the basis of requirement specification document.	*. this type of testing of software is started after detail design document.
*. No knowledge of programming is required.	*It is mandatory to have knowledge of programming.
*. It is the behavior testing of the software.	*. It is the logic testing of the software.
*. It is applicable to the higher levels of testing of software.	*. It is generally applicable to the lower levels of software testing.
*. It is also called closed testing.	*. It is also called as clear box testing.

*. It is least time consuming.	*. It is most time consuming.
*. It is not suitable or preferred for algorithm testing.	*. It is suitable for algorithm testing.
*. Can be done by trial and error ways and methods.	*. Data domains along with inner or internal boundaries can be better tested.
Ex: search something on google by using keywords.	Ex: by input to check and verify loops.

1)

2).

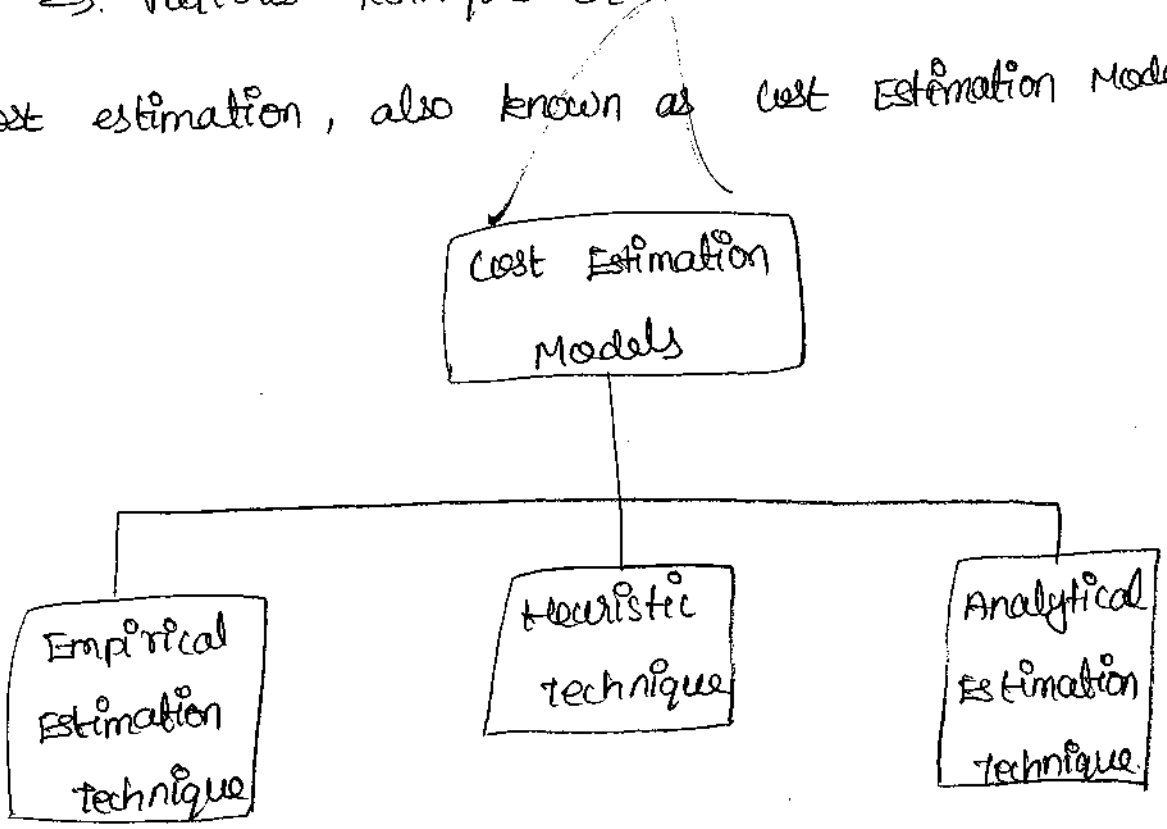
15.

b). The various software cost estimation techniques:

⇒. Cost estimation simply means a technique that is used to find out the cost estimates.

⇒. The cost estimate is the financial spend that is done on the efforts to develop and test software in Software Engineering.

⇒. various techniques or models are available for cost estimation, also known as cost Estimation Models.



Empirical Estimation Technique :

⇒. Empirical estimation is a technique or model in which empirically derived formulas are used for predicting the data that are a required and essential part of the software project planning step.

⇒. these techniques are usually based on the data that is collected previously from a project and also based on some guesses, prior experience with the development of similar types of projects, and assumptions.

⇒. It uses the size of the software to estimate the effort.

⇒. However, as there are many activities involved in empirical estimation techniques, this technique is formalized.

Heuristic technique :

⇒. Heuristic word is derived from a Greek word that means "to discover". The heuristic technique is a technique or model that is used for solving problems, learning or discovery in the practical methods which are used for achieving immediate goals.

⇒. these techniques are flexible and simple for taking quick decisions through shortcuts and good enough calculations, most probably when working with complex data.

⇒. But the decisions that are made using this technique are necessary to be optimal.

⇒. It is also used to increase or speed up the analysis and investment decisions.

Analytical Estimation Technique:

⇒. Analytical estimation is a type of technique that is used to measure work.

⇒. In this technique, firstly the task is divided or broken down into its basic component operations or elements for analyzing.

⇒. Second, if the standard time is available from some other source, then these sources are applied to each element or component of work.

⇒. Third, if there is no such time available, then the work is estimated based on the experience of the work.

Selvakumar R

190110

10.6.21

Software Engineering

60/100

CIA-3

13/6/21

Part-A

Answer all the Questions

(1) Software Engineering.

Systematic application of Engineering approaches to the development of software

2. The Available software risk

Internal risks

That are within the control of project manager

External risk

That are beyond the control of project manager.

3. The levels of phase design.

* Interface Design
* Architectural Design
* Detailed Design.

4.

Verification	Validation.
* The Verifying Process includes checking documents design, code and program	* It is a dynamic Mechanism of Testing * Validating the actual Product

5 Black box testing:-

* Demonstrate that software functions are optional
* That input is properly accepted and output is correctly Produced.
* Integrity of external information.

6 RFP

* Request for proposal
* Part of the bidding procedure for a product of service.
* To provide a structured way for companies.

control Structure Testing:-

* Control Structure Testing is a group of White-box Testing Methods => condition testing.

8 Cyclomatic complexity

The MSDN States: cyclomatic complexity Measures the number of linearity independent paths through the Method, which is determined by the number and complexity of conditional branches

$CC = E - N + 1$

9 COCOMO II Model.

COCOMO II is the revised version of the original COCOMO and its is developed at the University of Southern California

It is the Model that allows one to estimate the cost, effort and schedule when planning a new software development activity

10 The factor that leads to risk:

* The size of the scale
* The number of people who will be affected by the buying decision
* The length of life of the product

Part-B

11) a) The software process Model :-

=> A software process Model is an abstract representation of the development process

=> A models specify the stages and order of a process

=> So, think of this as a representation of the order of activities of the process and the sequence in which they are performed

=> The tasks to be performed

=> The input and output of each task

=> The post and post conditions for each task.

=> The flow and sequence of each task

=> The goal of a software process Model is to Provide guidance for controlling and coordinating the tasks to achieve the end product and objectives as effectively as possible

=> There are Many kinds of Process Models for Meeting different requirements

=) We refer to these as SDLC models

=) The most popular and important SDLC Models are as follows :-

* ~~Waterfall~~ Model
* V Model
* Incremental Model
* RAD Model
* Agile Model
* Iterative Model
* Prototype Model
* Spiral Model

=> choosing the right Software process model for four Projects can be difficult

=> If you know your requirements well, it will be easier to select a Model that best matches your needs

⇒ You need to keep the following factors in mind when selecting your software process

Project size:- Consider the size of the project you will be working on. Larger projects mean bigger teams, so you'll need more extensive and elaborate project management plans.

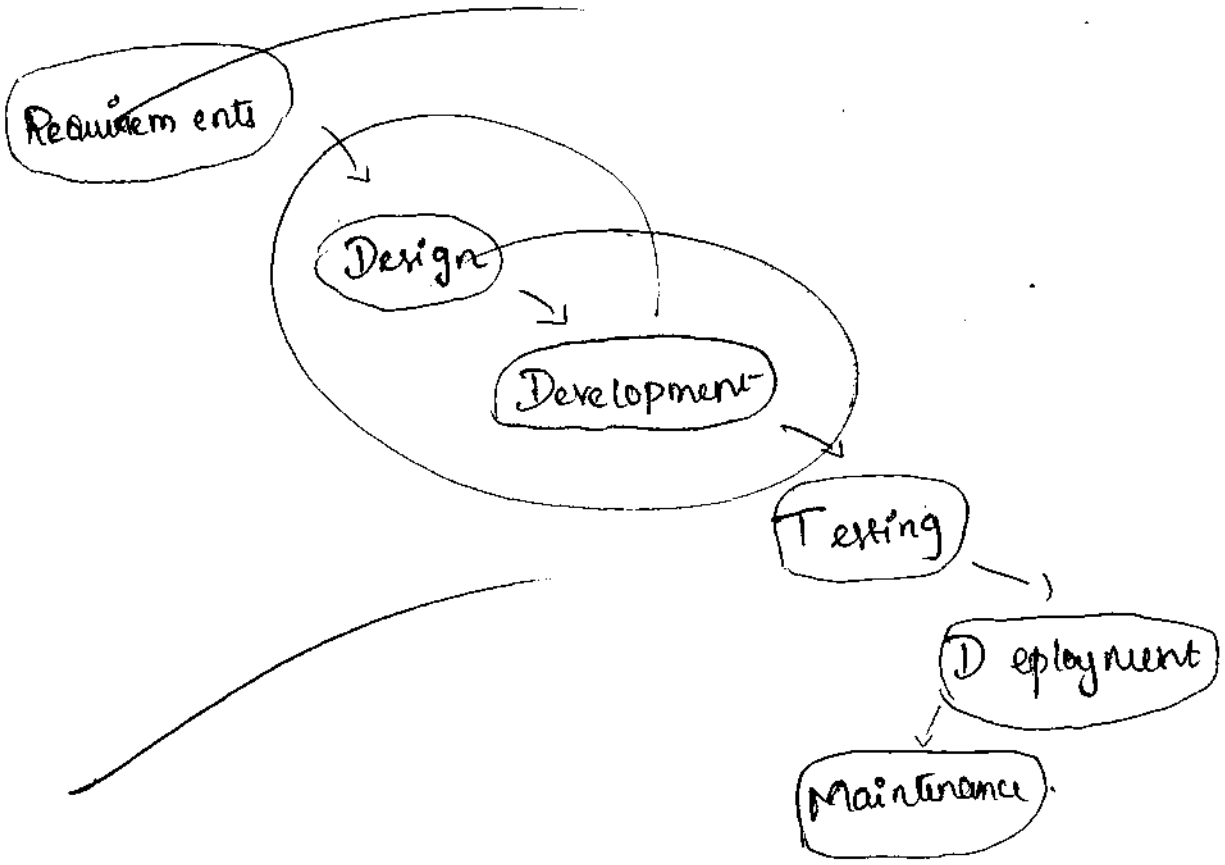
Project complexity :-

Complex projects may not have clear requirement The requirement may change often, and the cost of delay is high.

Cost of delay:- Is the project highly time-bound with a huge cost of delay.

Familiarity with technology:- This involves the developers knowledge and experience with the project domain, software tools, language and methods need for development

The waterfall model is a breakdown of Project activities into linear sequential phases - where each phase depends on the deliverables of the previous one and corresponds to a specialization of tasks.



Waterfall model.

12)

a) Requirement Engineering process:-

=) Requirement Engineering is the process of defining, documenting and maintaining requirements

=) It is like a process of gathering and defining servicing provided by the system.

=) Requirements Engineering Process consists of the following main activities.

* Requirements elicitation
* Requirements Specification
* Requirements Verification and validation
* Requirements Management.

Requirements elicitation :-

It is related to the various ways used to gain knowledge about the project domain and requirements.

⇒ The various source of domain knowledge include customers, business manuals, the existing software of same type, standards and other stakeholders of the project.

Requirements Specification :-

⇒ This activity is used to produce formal software required models.

⇒ All the requirements including the functional as well as the non-functional requirements and the constraints are specified by the models in totality

⇒ During specification, more knowledge about the problem may be required which can gain trigger the elicitation process

⇒ The models used at this stage include E R Diagrams, data flow diagrams.

Requirements Verification and Validation :-

=) It refers to the set of tasks that ensures that the software correctly implements a specific function.

=) It refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.

=) If requirements are not validated, errors in the requirement definitions would propogate to the successive stages resulting in a lot of modification and rework.

=) The main steps for this process include :-

* The requirements should be consistent with all other requirements.

* The requirements should be complete ; every other

* The requirements should be practically achievable

Reviews buddy checks, making test cases etc. are some of the methods used for this

Requirements Management :-

* Requirement management is the process of analying, documenting, tracking, prioritizing and agreeing the requirement and controlling the communication of relevant stakeholds

* Being able to modify the software as per requirements in a systematic and controlled manner is an extremely important part of the requirements engineering process.

13) b) Reconstruct the Taxonomy of Architectural Styles.

(1) Data Centered architecture.

A view is "a representation of a whole system from the perspective of a related set of concerns. Specification of the conventions of constructing and using a view.

A pattern or a template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis.

A word about presentation. To avoid having on single approach take all explanation space, we illustrate our classification with as many as different works as possible.

Data flow architecture

This term refers to the architecture that exists a human minds or in the software documentation In the literature Data flow architecture is also Qualified as idealized, intended, as designed or logical

we do not take in to accounts works like Archjava that extend traditional languages in to min architecture and programming element or other architecture) is not extracted from existing applications.

we also exclude approaches proposing general methodology or guidelines that do not stress a specific point to support software architecture re construction.

(3) Call and Return Architecture.

Software architecture reconstruction is a reverse enginuering approaches the aims at constructing viable architectural views of a software applications

The literature uses several other terms to refer to Call and Return Architecture.

Remote procedure call architecture:

This component is used to present in a main program or sub program architecture distributed among Multiple computers on a network

(4) Object oriented Architecture.

The components of a system encapsulate data and the operations that must be applied to manipulate the data. The coordination and communication between the components are established via the message passing.

Layered Architecture:-

A number of different layers are defined with each layer performing a well-defined set of operations. Each layer will do some operations that becomes closer to machine instruction set progressively.

At the outer layer components will receive the user interface operations and at the inner layers, components will perform the operating system interfacing

Intermediate layers to utility services and applications software functions.

11) a) (a) Data flow testing.

This testing technique emphasis to cover all the data variables included in the program.

It tests where the variables were declared and defined and where they were used or changed.

* Statements where variables receive values
* Statements where these values are used or referenced

Data flow Testing uses the control flow graph to find the situations that can interrupt the flow of the program. Reference or define anomalies in the flow of data are detected at the time of association between value and variables.

* A variable is defined but not used or referenced.
* A variable is used but never defined.
* A variable is defined twice before it is used.

Advantages of Data Flow Testing:-

To find a variable that is used but never defined.

To find a variable that is defined never used.

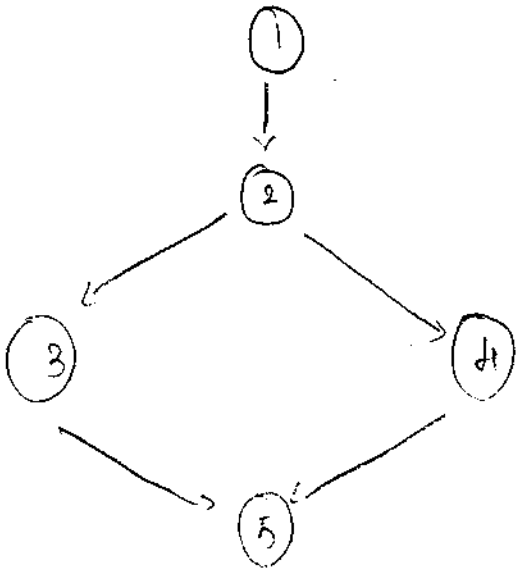
To find a variable that is defined multiple times before it is use.

Deallocating a variable before it is used.

Disadvantages of Data flow Testing:-

Time consuming and costly process

Requires knowledge of Programming languages.



(b) Integration testing.

Integration testing is type of testing meant to check the combinations of different Units, their interactions the way sub systems Unit into one common system and code compliance with the requirements.

Steps
=

Prepare the test integration plan

Decide on the type of integration testing approach

Design test cases, test scenario and test scripts accordingly.

Deploy the chosen modules together and get the integration tests running

Track the defects and record the test results of tests.

Types
=

(1) Big-Bang Integration testing

(2) Bottom up integration testing

(3) Top-Down Integration testing.

Mixed integration testing

Advantages:-

Seperately debugged Module

Few or no drivers needed

It is more stable and accurate at the aggregate level

Mixed approach is useful for very large projects having several sub projects.

Disadvantages:-

Needs many stubs

Module at lower level are tested inadequately

This integration testing cannot be used for smaller system with huge interdependence between different modules.

15) a Analyze the needs and activities of risk Management.

(1) Project Risks :-

These are the risks which affect the project schedule or resources.

Performance scope quality or technological risks.

* the project when completed fails to perform as intended or fails to meet the mission requirements that generated the justification for the project

Environment, safety and health risks

- these are the detrimental effects on the environment or that hidden hazards.

* Schedule risks. Projects take longer schedules

* cost risks -

- the project costs more than budgeted

* loss of support :-

loss of Public stake holder support for the project goals and objectives may utlized lead to the reduction of scope and or funding cuts.

Product risks:-

Product risk is the possibility that the system or software might fail to satisfy or fulfill some reasonable expectation of the customer, user or stake holder.

These are also called as Quality risks as they are risks to the Quality of the product.

The Big Risks

~~Value Risks~~
Usability Risks
feasibility Risks.

Management

Generally, risk Management entails risk containment and mitigation
When it comes to product Management, you should always be ready to act whenever a risk arises.
In the realm of Product Management, risk can be defined as the possibility of the project suffering loss.

(3) Business Risks :-

The term business risks refers to the possibility of a commercial business making inadequate profits due to uncertainties

for example :- changes in taxes, changing preferences of consumers, strikes increased competition, changes in government policy, obsolescence etc.

Identify Business Risks

(1) Analyse the sources that may trigger problems
(2) Act now
(3) involve employees
(4) Make a list of industry - specific risks
(5) Create a record of risks.

Types of Risks.

(1) Strategic risks
(2) compliance risks
(3) Financial risks
4 Operational risks

Causes of Business Risks.

(1) Natural causes
(2) Human causes
(3) Economic causes.

M. Praveen Kumar
10/5/21

CIA - III

PART-A

14/100

13/8/21

1. Software Engineering:

Systematic application of engineering approaches to the development of software.

2. Software risk:

Internal risks that are within the control of the project manager and

External risks that are beyond the control of the project manager

3. Phases of design

* Interface Design
* Architectural Design
* Detailed Design

4.

Verification	Validation.
* The verifying process includes checking documents, design, code, and program	* It is a dynamic mechanism of testing * Validating the actual product

5. Black box testing:

- Demonstrate that software function are operational
- That input is properly accepted and output is correctly produced
- Integrity of external information.

6 State RFP

- Request for proposal
- Part of the bidding procedure for a product or service
- To provide a structured way for companies

7 Define control structure testing:

Control structure testing is a group of white-box testing methods. CONDITION TESTING. It is a test case design method. It works on logical conditions in program module. It involves testing of both relational expressions and arithmetic expressions.

8 How to compute the cyclomatic complexity?

The MSDN states: "Cyclomatic complexity measures the number of linearly independent paths through the method, which is determined by the number and complexity of conditional branches.

Hence how cyclomatic complexity is calculated:

$CC = E - N + 1$.

9 Reproduce COCOMO II Model

COCOMO-II is the revised version of the original Cocomo (Constructive Cost Model) and is developed at University of Southern California. It is the model that allows one to estimate the cost, effort and schedule when planning a new software development activity.

10 List the factors that lead to Risk?

The size of the sale.

The number of people who will be affected by the buying decision.

The length of life of the product.

The customer's unfamiliarity with you, your company, and your product or services.

PART-B

11) The software Process Model:

* A software process model is a simplified representation of a software process.

* The software development process is complicated and involves a lot more than technical knowledge.

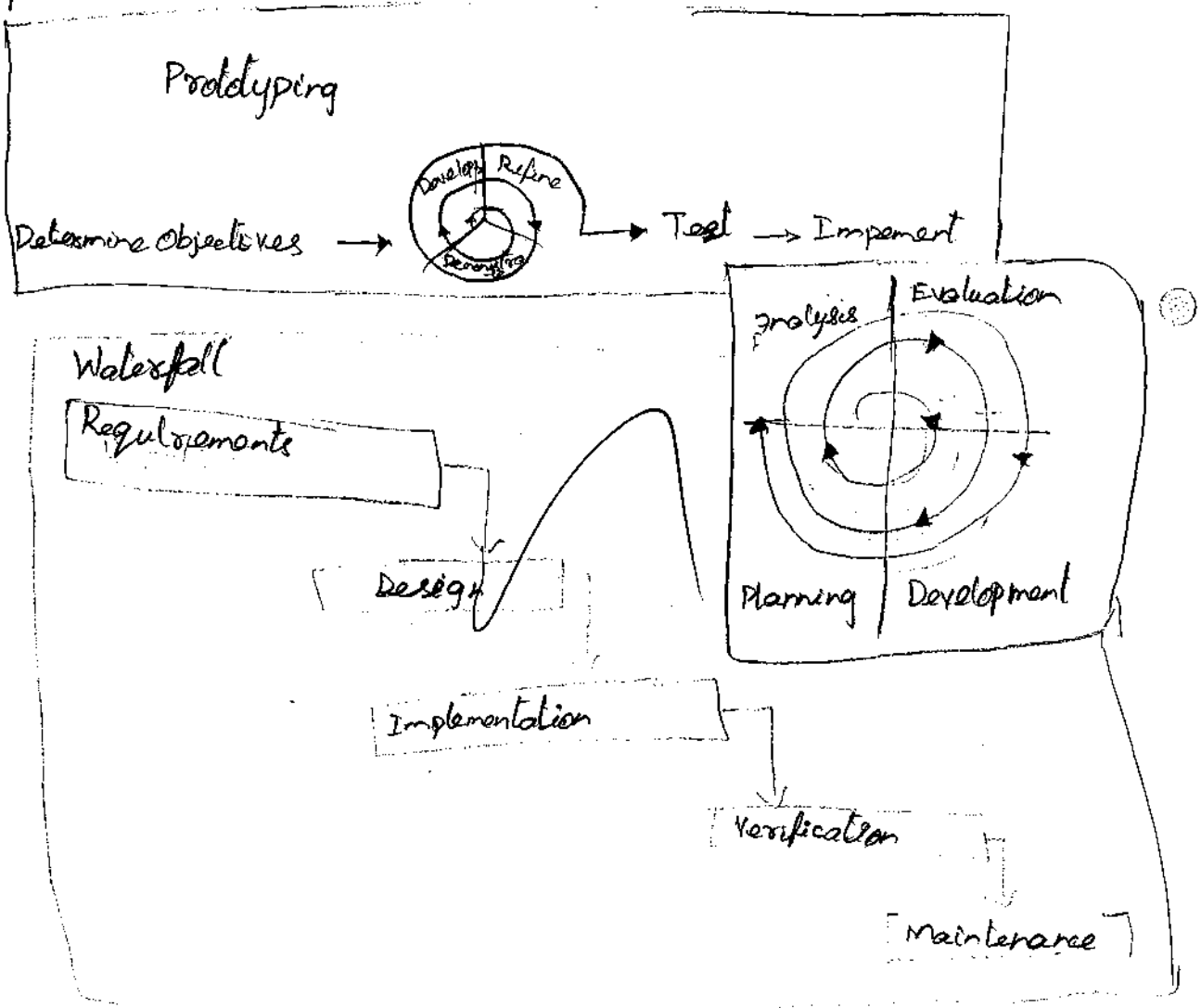
* A software process model is a abstract representation of the development process.

* A software process model is an abstraction of the software development process. The models specify the stages and order of a process.

* The order of activities of the process and the sequence in which they are performed.

* The task to be performed
* The input and output of each task
* The pre and post conditions for each task
* The flow and sequence of each task.

The goal of a software process model is to provide guidance for controlling and coordinating the tasks to achieve the desired end product and objectives as effectively as possible.



Source: Omar Elgabry

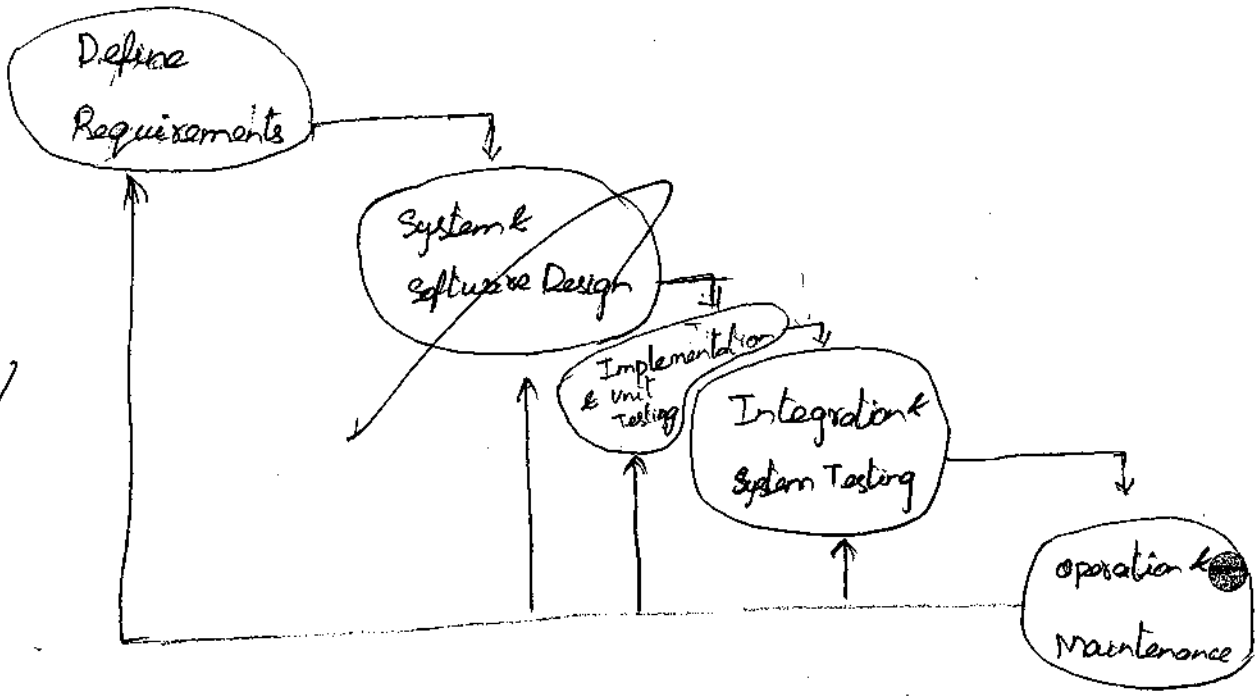
There are many kinds of process many kinds of Process models for meeting different requirements. We refer to these as SDLC models (Software Development Life Cycle models). The most popular and important SDLC models are as

- Waterfall model
- V model
- Incremental model
- RAD model
- Agile model
- Iterative model
- Prototype model
- Spiral model.

Waterfall Model

The Waterfall model is a sequential, plan driven-process where you must plan and schedule all your activities before starting the project. Each activity in the waterfall model is represented as a separate phase arranged in linear orders.

- Requirements
- Design
- Implementation
- Testing
- Deployment
- Maintenance.

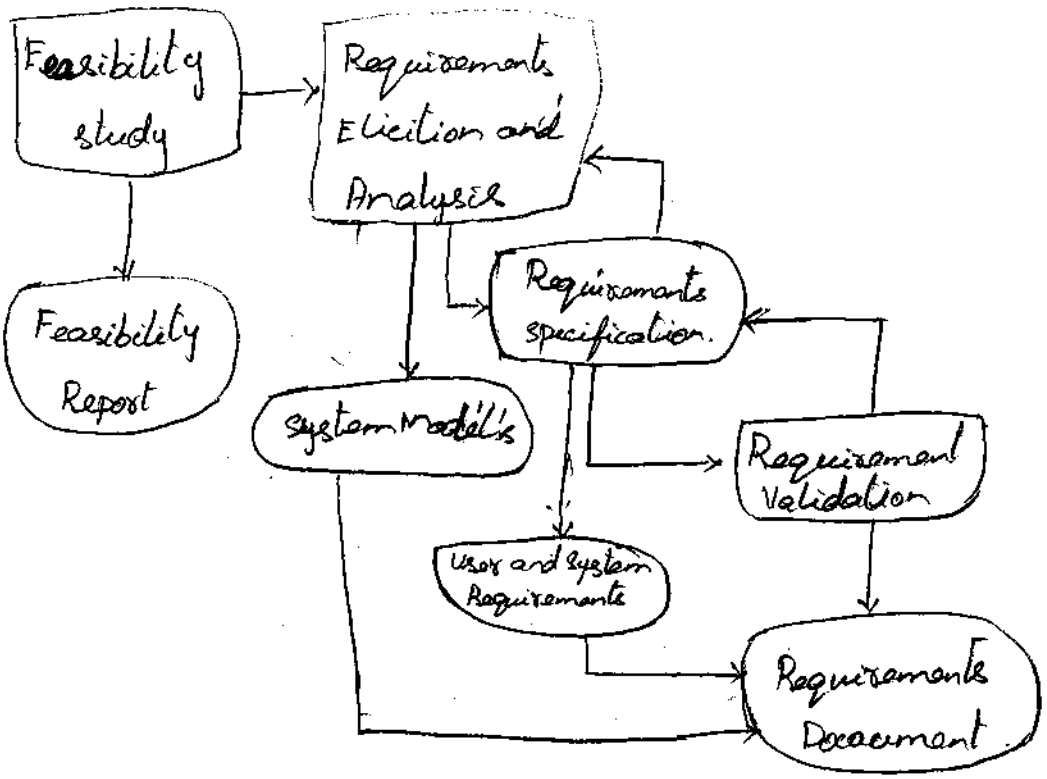


12/a) Demonstrate requirement engineering and its process.

*The process to gather the software requirements from clients analyze and document them is known as requirement engineering

Requirement Engineering Process

- Feasibility Study
- Requirement Elicitation and Analysis
- Software Requirement Specification
- Software Requirement Validation
- Software Requirement Management



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

* Technical Feasibility - Technical feasibility evaluates the current technologies which are needed to accomplish customer requirements within the time and budget.

* Operational Feasibility - Operational feasibility assesses the range on which the required software performs a series of tasks to solve business problem and customer requirements.

♦ Economic Feasibility - Economic feasibility decides whether the necessary software can generate financial profits for an organization.

• Requirement Elicitation and Analysis:

*This is also known as the gathering of requirements. Here requirements are identified with the help of customer and existing system 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 relationship and 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.

13. a) User interface design

The visual part of a computer application or operating system through which a client interacts with a computer or software. It determines how commands are given to the computer or the computer or the program and how data is displayed in the screen.

Types of User Interface:

There are two main types of User Interface.

- Text-Based User Interface or Command Line Interface.
- Graphical User Interface (GUI)

Text-Based User Interface. This method relies primarily on keyboard. A typical example of this is UNIX.

Advantages:

- Many and easier to customizations options
- Typically capable of more important tasks

Disadvantages:

- Relies heavily on recall rather than recognition
- Navigation is often more difficult

Graphical User Interface (GUI): GUI relies much more heavily on the mouse A typical example of this type of interface is any versions of the Windows operating system.

GUI characteristics

characteristics	Descriptions.
Windows	Multiple windows allow different information to be displayed simultaneously on the users screen
Icons	Icons different types of information on some system icons represent files on other icons describes process
Menu	Commands are selet from a menu rather than typed on a command language.
Pointing	A pointing device such as a mouse is used for selecting choices from a menu or indicating items of interest in a window
Graphics	Graphics elements can be mixed with text on the same display

Advantage:

- Less expert knowledge is required to use it
- Easier to Navigate and can look through folders quickly in a guess and check manner.
- The user may switch quickly from one task to another and can interact with several different applications

Disadvantage:

- Typically decreased options
- Usually less customizable Not easy to use one button for tons of different variations.

4a) Re/Write short notes on Data flow Testing and Integration testing

Data flow testing

Data flow testing is used to analyze the flow of data in the program. It is the process of collecting information about how the variable flow the data in the program It tries to obtain particular information of each particular point in the process

Data flow testing is a group of testing strategies to examine the control flow of programs in order to explore the sequence of variables according to the sequence of events It mainly focuses on the points at which values assigned to the variables and

the point at which these values are used by concentrating on both points. data flow can be tested.

Data flow testing uses the control flow graph to detect illogical things that can interrupt the flow of data Anomalies in the flow of data are detected at the time of associations between values and variables due to

- If the variables are used without initialization
- If the initialization are not used at least once

1. read x;	
2. If (x>0)	(1,2,t), (1,(2,f),x)
3. a=x+1	(1,3,x)
4. if (x<=0){	(1,(4,t),x), (1,(4,f),x)
5. if (x<1)	(1,(5,t),x), (1,(5,f),x)
6. x=x+1; (goto 5)	(1,6,x)
else	
7. a=x+1	(7,7,x)
8. print a;	(6,(5,f)x), (6,(5,t)x)
	(6,6,x)
	(3,8,a), (7,8,a).

In this

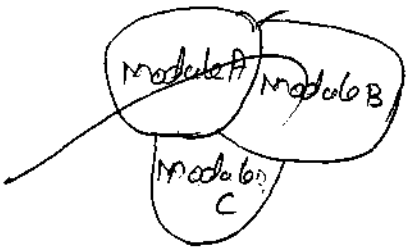
Integration testing

Integration testing is the second level of the software testing process comes after unit testing. In this testing units or individual components of the software are tested on a group. The focus of the integration testing level is to expose defects at

the form of interaction between integrated components or units.

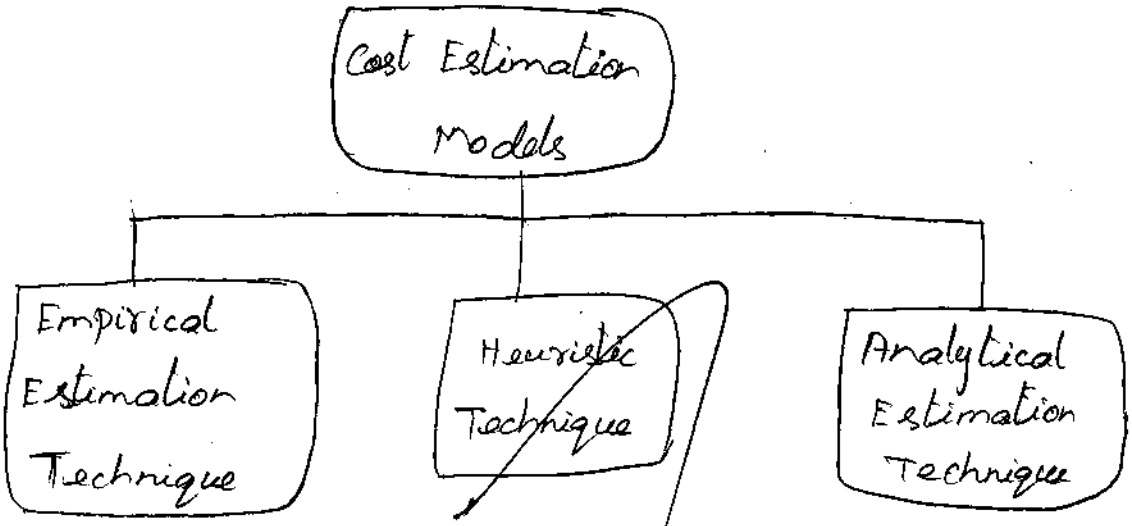
Unit testing uses modules for testing purpose and these modules are combined and tested in integration testing. The software is developed with a number of software modules that are coded by different coders or programmers. The goal of integration testing is to check the correctness of communication among all the modules.

Tested in Unit Testing



Once all the components or modules are working independently then we need to check the data flow between the dependent modules is known as integration testing.

15. Software Cost Estimation



For any new software project it is necessary to know how much it will cost to develop and how much development time will it take. These estimates are needed before development is initiated but how is this done? Several estimation procedures have been developed and are having the following attributes in common.

* Project scope must be established in advanced.
* Software metrics are used as a support from which evaluation is made.
* The project is broken onto small PCs which are which are estimated individually

To achieve true cost & schedule estimate several option arise.

* Delay estimation
* Used symbol decomposition techniques to generate.

project cost and schedule estimates

* Require one or more automated estimates tools.

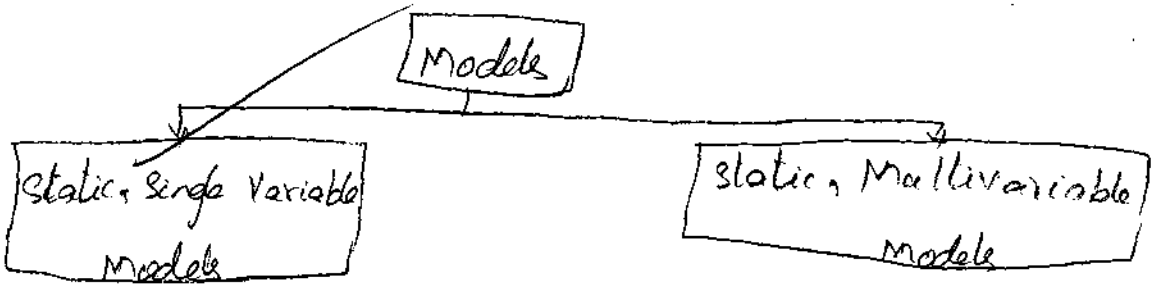
Uses of cost Estimation:

*During the planning stage one needs to choose how many engineers are required for the project and to develop a schedule

* In monitoring the projects progress one needs to access whether the project is progressing according to the procedure and takes corrective action if necessary

Cost Estimation Models.

A models may be static or dynamic In a static model a single variable is taken as a key element for calculating cost and time In a dynamic model all variable are interdependent and there is one basic variable.



static single variable Models. When a model makes use of single variables to calculate desired values such as cost time efforts etc is said to be a single variable model The most

common equation is

$$C = aL^b$$

Where C = Costs

L = Size

a and b are constants.

The software Engineering Laboratory. software production.

$$E = K L^{0.93}$$

$$DOC = 30.4 L^{0.90}$$

$$D = 4.6 L^{0.26}$$

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu.

Estd. 2000

Department of Computer Science and Engineering

Useful Website / E-Content Details - Academic Year (2020-21)

Course Code & Course Name : **19CSC12/SOFTWARE ENGINEERING**

Name of the Faculty : **V.Karuppuchamy**

Year/Sem/Sec : **II/IV/B**

UNIT-I : SOFTWARE PROCESS AND PROJECT MANAGEMENT

S.No	Topic Name	URL Link
1	**Software Process**	https://www.youtube.com/watch?v=YMbAdgb6pG8
2	**Software Project Management**	https://www.youtube.com/watch?v=TYBVAvWkG6M
3	**Estimation – LOC ,FP Based Estimation**	https://www.youtube.com/watch?v=nN379biPGxE
4	**COCOMO Model**	https://www.youtube.com/watch?v=6ySF9wNDAZo
5	**Project Scheduling**	https://www.apm.org.uk/resources/what-is-project-management/what-is-scheduling-in-project-management/

UNIT-II : REQUIREMENTS ANALYSIS AND SPECIFICATION

S.No	Topic Name	URL Link
1	**Software Requirements**	https://blog.bit.ai/software-requirements-document/
2	**Functional and Non-Functional**	https://www.youtube.com/watch?v=j4WITZFLkUM
3	**User requirements**	https://www.coursera.org/lecture/client-needs-and-software-requirements/3-1-2a-user-requirements-fU9RE
4	**Classical analysis**	https://www.youtube.com/watch?v=lHA9hBaf-EU
5	**Structured system Analysis**	https://www.youtube.com/watch?v=69PIhEvYmm8

UNIT-III : SOFTWARE DESIGN

S.No	Topic Name	URL Link
1	**Design Concepts**	https://www.delve.com/design-concepts
2	**Design Model**	https://study.com/academy/lesson/design-model-in-software-engineering-elements-examples.html
3	**Design Heuristic**	https://www.nngroup.com/articles/ten-usability-heuristics/
4	**Architectural styles**	https://www.coursera.org/lecture/engineering-practices-secure-software-quality/architectural-styles-DnFzH
5	**Architectural Design**	https://www.youtube.com/watch?v=ly8orBNiNQM

UNIT-IV : TESTING AND IMPLEMENTATION

S.No	Topic Name	URL Link
1	**Basis Path Testing**	https://www.youtube.com/watch?v=TAFhCV721tY
2	**Control Structure Testing**	https://www.youtube.com/watch?v=L3O92meOzXA
3	**Black Box Testing**	https://www.guru99.com/black-box-testing.html
4	**Regression Testing**	https://www.guru99.com/regression-testing.html
5	**cyclomatic complexity**	https://www.youtube.com/watch?v=HRx8IOJF2H0

UNIT-V : PROJECT MANAGEMENT

S.No	Topic Name	URL Link
1	**Make/Buy Decision**	https://www.youtube.com/watch?v=vxZSaDtQgrs
2	**Planning – Project Plan**	https://www.projectmanager.com/training/project-planning-for-beginners
3	**Planning Process**	https://www.youtube.com/watch?v=Do8iykQKMfU
4	**Risk Management**	https://searchcompliance.techtarget.com/definition/riskmanagement
5	**RMMM**	https://www.youtube.com/watch?v=LD9WpiggFAs

17/2/2024

Course Faculty

HoD

MUTHAYAMMAL ENGINEERING COLLEGE

(An Autonomous Institution)

(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)

Rasipuram - 637 408, Namakkal Dist., Tamil Nadu

Estd. 2000

IQAC

MUST KNOW CONCEPTS

MKC

CSE

2020-21

Course Code & Course Name : 19CSC12/SOFTWARE ENGINEERING

Year/Sem/Sec : II/IV/A&B

S.No.	Term	Notation (Symbol)	Concept / Definition / Meaning / Units / Equation / Expression	Units
UNIT I : SOFTWARE PROCESS AND AGILE DEVELOPMENT				
1.	**Software**		Software is a program that enables a computer to perform a specific task, as opposed to the physical components of the system	
2.	**Software Engineering**		Software engineering is the systematic application of engineering approaches to the development of software.	
3.	**Software Process**		Software process (also known as software methodology) is a set of related activities that leads to the production of the software.	
4.	**Waterfall Model**		The waterfall model is a sequential approach, where each fundamental activity of a process represented as a separate phase, arranged in linear order.	
5.	**Prototype**		A prototype is useful when a customer or developer is not sure of the requirements, or of algorithms, efficiency, business rules, response time, etc.	
6.	**Spiral Model**		The spiral model is a risk-driven where the process is represented as spiral rather than a sequence of activities.	
7.	**Framework Activities**		1. Communication 2. Planning 3. Modelling 4. Construction 5. Deployment	
8.	**Types of prescriptive process models**		1. The Waterfall Model 2. Incremental Process model 3. Evolutionary Process Models 4. Concurrent Models	
9.	**Incremental Process Model**		The incremental model combines the elements of waterfall model and they are applied in an iterative fashion.	
10.	**Evolutionary Model**		The Evolutionary development model divides the development cycle into smaller, incremental waterfall models in which users are able to get access to the product at the end of each cycle.	
11.	**Concurrent Process Model**		The concurrent process model defines a series of events that will trigger transitions from state to state for each of the software engineering activities.	

12.	**Special process models**		Special process models take many features from one or more conventional models.	
13.	**Component based development**		The component based development model incorporates many of the characteristics of the spiral model.	
14.	**Aspect oriented Software Development**		Aspect Oriented Software Development (AOSD) often referred to as aspect oriented programming (AOP), a relatively new paradigm that provides process and methodology for defining, specifying designing and constructing aspects.	
15.	**Software Project Management**		Software Project Management (SPM) is a proper way of planning and leading software projects. It is a part of project management in which software projects are planned, implemented, monitored and controlled.	
16.	**Management Spectrum**		Effective software project management focuses on the four P's: people, product, process, and project.	
17.	**Project estimation Types**		Software size estimation Effort estimation Time estimation Cost estimation	
18.	**Project size estimation techniques**		Lines of Code Number of entities in ER diagram Total number of processes in detailed data flow diagram Function points	
19.	**Lines of Code (LOC):**		As the name suggest, LOC count the total number of lines of source code in a project.	
20.	**units of LOC**		KLOC- Thousand lines of code NLOC- Non comment lines of code KDSI- Thousands of delivered source instruction	
21.	**Function Point Analysis**		In this method, the number and type of functions supported by the software are utilized to find FPC(function point count).	
22.	**AGILE**		Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements.	
23.	**Unified Process Model**		A software process that is: • use-case driven • architecture-centric • iterative and incremental	
24.	**Extreme Programming**	**XP**	Extreme Programming (XP) is an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team.	
25.	**Project-task scheduling**		Project-task scheduling is a significant project planning activity. It comprises deciding which functions would be taken up when.	
UNIT II : REQUIREMENTS ANALYSIS AND SPECIFICATION				
26.	**Requirement Engineering**		The process to gather the software requirements from client, analyze and document them is known as requirement engineering.	
27.	**Software requirements**		Software requirements is a field within software engineering that deals with establishing the needs of stakeholders that are to be solved by software	

28.	**User requirements**		User requirements, often referred to as user needs, describe what the user does with the system, such as what activities that users must be able to perform.	
29.	**System requirements**		System requirements are the configuration that a system must have in order for a hardware or software application to run smoothly and efficiently.	
30.	**Hardware system requirements**		Hardware system requirements often specify the operating system version, processor type, memory size, available disk space and additional peripherals, if any, needed.	
31.	**Software system requirements**		Software system requirements, in addition to the aforementioned requirements, may also specify additional software dependencies (e.g., libraries, driver version, framework version).	
32.	**System Requirement Document (SRD)**		The System Requirement Document (SRD) defines system level functional and performance requirements for a system. It should include a system level description of all software elements required by the preferred system concept.	
33.	**Requirement Engineering Process**		Feasibility Study Requirement Elicitation and Analysis Software Requirement Specification Software Requirement Validation Software Requirement Management	
34.	**Types of Feasibility**		Technical Feasibility Operational Feasibility Economic Feasibility	
35.	**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.	
36.	**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.	
37.	**Software Requirement Management**		Requirement management is the process of managing changing requirements during the requirements engineering process and system development.	
38.	**Classical Analysis**		The evaluation of an activity to identify its desired objectives and determine procedures for efficiently attaining them.	
39.	**Structured Analysis Tools**		Data Flow Diagrams Data Dictionary Decision Trees Decision Tables Structured English Pseudocode	
40.	**Data Flow Diagrams**		DFD is easy to understand and quite effective when the required design is not clear and the user wants a notational language for communication.	
41.	**Context Diagram**		A context diagram helps in understanding the entire system by one DFD which gives the overview of a	

			system.	
42.	**Data Dictionary**		A data dictionary is a structured repository of data elements in the system. It stores the descriptions of all DFD data elements that is, details and definitions of data flows, data stores, data stored in data stores, and the processes.	
43.	**Decision Trees**		Decision trees are a method for defining complex relationships by describing decisions and avoiding the problems in communication.	
44.	**Decision Tables**		Decision tables are a method of describing the complex logical relationship in a precise manner which is easily understandable.	
45.	**Structured English**		Structure English is derived from structured programming language which gives more understandable and precise description of process.	
46.	**Pseudocode**		A pseudocode does not conform to any programming language and expresses logic in plain English.	
47.	**Technical Feasibility**		Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.	
48.	**Functional Requirement**		A functional requirement defines a system or its component. It describes the functions software must perform.	
49.	**Non-Functional Requirement**		A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system. Failing to meet non-functional requirements can result in systems that fail to satisfy user needs.	
50.	**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.	
UNIT III : SOFTWARE DESIGN				
51.	**Software design**		Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.	
52.	**Objectives of Software Design**		Correctness: Efficiency: Understandability Completeness: Maintainability	
53.	**Levels Of Phases Of Design**		Interface Design Architectural Design Detailed Design	
54.	**Interface design**		Interface design is the specification of the interaction between a system and its environment. This phase proceeds at a high level of abstraction with respect to the inner workings of the system.	
55.	**Architectural design**		Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them.	
56.	**Issues in architectural design**		Gross decomposition of the systems into major components. Allocation of functional responsibilities to components.	

			Component Interfaces	
57.	**Detailed Design**		Design is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures.	
58.	**Software Design Concepts**		The software design concept simply means the idea or principle behind the design. It describes how you plan to solve the problem of designing software, the logic, or thinking behind how you will design software.	
59.	**Design Heuristic**		Heuristics refers to a non-optimal solution for experience-based techniques to solve problems, learning, and discovery. The main goal of heuristic evaluations is to identify any problems associated with the design of user interfaces.	
60.	**Architectural design**		Architectural design is a process for identifying the sub-systems making up a system and the framework for sub-system control and communication. The output of this design process is a description of the software architecture.	
61.	**Architectural styles**		The software that is built for computer-based systems can exhibit one of these many architectural styles.	
62.	**Data flow architectures**		This kind of architecture is used when input data to be transformed into output data through a series of computational manipulative components.	
63.	**User Interface Design**		User interface is the front-end application view to which user interacts in order to use the software.	
64.	**Command Line Interface**		Command Line Interface provides a command prompt, where the user types the command and feeds to the system.	
65.	**Graphical User Interface**		Graphical User Interface provides the simple interactive interface to interact with the system.	
66.	**Abstraction**		Abstraction simply means to hide the details to reduce complexity and increases efficiency or quality.	
67.	**Modularity**		Modularity in design means to subdivide a system into smaller parts so that these parts can be created independently and then use these parts in different systems to perform different functions.	
68.	**Refinement**		Refinement simply means to refine something to remove any impurities if present and increase the quality.	
69.	**Pattern**		The pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern.	
70.	**Refactoring**		Refactoring simply means to reconstruct something in such a way that it does not affect the behaviour or any other features.	
71.	**Two levels of abstraction**		Architecture in the small Architecture in the large	
72.	**User Interface Golden rules**		Strive for consistency - Consistent sequences of actions should be required in similar situations. Identical terminology should be used in prompts, menus, and help screens. Consistent commands should be employed throughout.	

73.	**Traditional Components**		Traditional components are designed based on different constructs like. Sequence implements processing steps that are essential in the specification of any algorithm.	
74.	**Interface Validation**		This phase focuses on testing the interface. The interface should be in such a way that it should be able to perform tasks correctly and it should be able to handle a variety of tasks.	
75.	**Information Hiding**		Information hiding simply means to hide the information so that it cannot be accessed by an unwanted party.	
Unit-IV : TESTING AND IMPLEMENTATION				
76.	**Software Testing**		Software Testing is vital for any software development life cycle, it is fundamental to ensure the software quality and to have a workable functional software at the end of the project.	
77.	**White-box testing**		It is conducted to test program and its implementation, in order to improve code efficiency or structure. It is also known as 'Structural' testing.	
78.	**White-box testing techniques**		Control-flow testing Data-flow testing	
79.	**Basic Path Testing**		Path Testing is a method that is used to design the test cases. In path testing method, the control flow graph of a program is designed to find a set of linearly independent paths of execution.	
80.	**Advantages of Path Testing**		Path testing method reduces the redundant tests. Path testing focuses on the logic of the programs. Path testing is used in test case design.	
81.	**Control structure testing**		Control structure testing is used to increase the coverage area by testing various control structures present in the program.	
82.	**Condition Testing**		Condition testing is a test cased design method, which ensures that the logical condition and decision statements are free from errors.	
83.	**Loop Testing**		Loop testing is actually a white box testing technique. It specifically focuses on the validity of loop construction.	
84.	**Concatenated Loops**		If loops are not dependent on each other, contact loops can be tested using the approach used in simple loops. if the loops are interdependent, the steps are followed in nested loops	
85.	**Black box testing**		Black box testing is a type of software testing in which the functionality of the software is not known. The testing is done without the internal knowledge of the products.	
86.	**Regression Testing**		Regression Testing is the process of testing the modified parts of the code and the parts that might get affected due to the modifications to ensure that no new errors have been introduced in the software after the modifications have been made.	
87.	**Advantages of Regression Testing**		It ensures that no new bugs have been introduced after adding new functionalities to the system.	
88.	**Disadvantages of Regression Testing**		It can be time and resource consuming if automated tools are not used. It is required even after very small changes in the code.	

89.	**Unit testing**		Unit testing, a testing technique using which individual modules are tested to determine if there are any issues by the developer himself. It is concerned with functional correctness of the standalone modules.	
90.	**Integration testing**		Integration testing is the process of testing the interface between two software units or module. It's focus on determining the correctness of the interface.	
91.	**Bottom-Up Integration Testing**		In bottom-up testing, each module at lower levels is tested with higher modules until all modules are tested.	
92.	**Top-down integration testing**		Top-down integration testing technique used in order to simulate the behaviour of the lower-level modules that are not yet integrated.	
93.	**System Testing**		System Testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements.	
94.	**Performance Testing**		Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.	
95.	**Top-Down Integration Testing Advantages**		Separately debugged module. Few or no drivers needed. It is more stable and accurate at the aggregate level	
96.	**Top-Down Integration Testing Disadvantages**		Needs many Stubs. Modules at lower level are tested inadequately.	
97.	**Mixed Integration Testing Disadvantages**		For mixed integration testing, require very high cost because one part has Top-down approach while another part has bottom-up approach.	
98.	**Load Testing**		Load Testing is a type of software Testing which is carried out to determine the behaviour of a system or software product under extreme load.	
99.	**Stress Testing**		Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.	
100.	**Scalability Testing**		Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.	
Unit-V : PROJECT MANAGEMENT				
101.	**Project management**		Project management is the application of processes, methods, skills, knowledge and experience to achieve specific project objectives according to the project acceptance criteria within agreed parameters.	
102.	**Make or buy decision**		Make or buy decision is always a valid concept in business. No organization should attempt to make something by their own, when they stand the opportunity to buy the same for much less price.	
103.	**Reasons for Making**		Cost concerns Desire to expand the manufacturing focus Need of direct control over the product	
104.	**COCOMO II Model**		COCOMO-II is the model that allows one to estimate the cost, effort and schedule when planning a new software development activity.	

105.	**Project planning**		Project planning is an organized and integrated management process, which focuses on activities required for successful completion of the project.	
106.	**Project Planning Process**		The project planning process involves a set of interrelated activities followed in an orderly manner to implement user requirements in software and includes the description of a series of project planning activities and individual(s) responsible for performing these activities.	
107.	**Objectives and scope of the project**		Techniques used to perform project planning Effort (in time) of individuals involved in project Project schedule and milestones Resources required for the project Risks associated with the project.	
108.	**Project Plan**		A project plan helps a project manager to understand, monitor, and control the development of software project. This plan is used as a means of communication between the users and project management team.	
109.	**RFP**		An RFP stands for "request for proposal" and is generated as part of the bidding procedure for a product or service. The purpose of an RFP is to provide a structured way for companies to learn about doing business with software development teams.	
110.	**Risk management**		Risk management is a management specialism aiming to reduce different risks related to a preselected domain to the level accepted by society. It may refer to numerous types of threats caused by environment, technology, humans, organizations and politics.	
111.	**Task**		**Task** is part of a set of actions which accomplish a job, problem or assignment.	
112.	**Management process**		Management process is a process of planning and controlling the performance or execution of any type of activity.	
113.	**Process**		Process is an ongoing collection of activities, with an inputs, outputs and the energy required to transform inputs to outputs.	
114.	**Task analysis**		Task analysis is the analysis or a breakdown of exactly how a task is accomplished, such as what sub-tasks are required	
115.	**Project manager**		Professional in the field of project management. Project managers can have the responsibility of the planning, execution, and closing of any project, typically relating to construction industry, architecture, computer networking, telecommunications or software development.	
116.	**Resources**		Resources are what is required to carry out a project's tasks. They can be people, equipment, facilities, funding, or anything else capable of definition (usually other than labor) required for the completion of a project activity.	
117.	**Allocation**		Allocation is the assignment of available resources in an economic way.	
118.	**Project network**		Project network is a graph (flow chart) depicting the sequence in which a project's terminal elements are to be completed by showing terminal elements and their dependencies.	

119.	**Quality, Cost, Delivery (QCD)**		Quality, Cost, Delivery (QCD) as used in lean manufacturing measures a business's activity and develops Key performance indicators. QCD analysis often forms a part of continuous improvement programs	
120.	**Scope**		Scope of a project in project management is the sum total of all of its products and their requirements or features.	
121.	**Six Sigma**		Six Sigma is a business management strategy, originally developed by Motorola, that today enjoys widespread application in many sectors of industry.	
122.	**Case study**		Case study is a research method which involves an in-depth, longitudinal examination of a single instance or event: a case. They provide a systematic way of looking at events, collecting data, analyzing information, and reporting the results.	
123.	**Portfolio**		Portfolio in finance is an appropriate mix of or collection of investments held by an institution or a private individual.	
124.	**Project**		Project : A temporary endeavor undertaken to create a unique product, service, or result.	
125.	**Constructive Cost Model**	**COCOMO**	Cocomo is a regression model based on LOC, i.e number of Lines of Code.	
			Placement Questions	
126.	**What is the average of first five multiples of 10?**		Average = $10*(1+2+3+4+5) * \frac{1}{5}$ $= 10* 15* \frac{1}{5}$ $= 10 * 3= 30$	
127.	**What is the difference in the place value of 5 in the numeral 754853?**		The digit 5 has two place values in the numeral, $5 * 10^5 = 50,000$ and $5 * 10^1 = 50$. ∴Required difference = 50000 - 50 = 49950	
128.	**A number added to 1459 so that it is exactly divisible by 12.**		On dividing 1459 by 12, the remainder is 7. ∴The number to be added would be = 12 - 7 = 5	
129.	**In the given expression (1.05)2 *x = 44.1, find the value of x.**		(1.05)2 *x=44.1 Or, x = 44.1/ (1.05)2 = 44.1/ (1.05 * 1.05) Hence, x = 40.00	
130.	**If January 1, 1996, was Monday, what day of the week was January 1, 1997?**		The year 1996 is divisible by 4, so it is a leap year with 2 odd days. As per the question, the first day of the year 1996 was Monday, so the first day of the year 1997 must be two days after Monday. So, it was Wednesday.	
131.	**A: B: C is in the ratio of 3: 2: 5. How much money will C get out of Rs 1280?**		C's share = [C's ratio/ sum of ratios] * total amount C's share = (5/10) * 1280 C's share = 640	
132.	**Today is Wednesday, after 69 days, it will be**		Each day of a week is repeated after 7 days, so after 70 days, it will be Wednesday. Therefore, after 69 days, it will be Tuesday.	
133.	**A Number times the hands of a clock coincide in a day**		The hands of a clock coincide only once between 11 O' clock and 1 O' clock, so in every 12 hours, the hands of a clock will coincide for 11 times. ∴ In a day or 24 hours, the hands of a clock will coincide	

MUTHAYAMMAL ENGINEERING COLLEGE
(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu
Estd. 2000
IQAC

Online Exam Question's With Answer

Course Code & Course Name : 19CSC12/SOFTWARE ENGINEERING
Year/Sem/Sec : II/IV/A

Sl. No.	Question	Option A	Option B	Option C	Option D	Correct option
1	Software is considered to be collection of	Programming code	Associated libraries	Documentations	All of the above	d
2	The process of developing a software product using software engineering principles and methods is referred to as	Software Engineering	System Models	Software Models	software Evolution	b
3	Lehman has given laws for software evolution and he divided the software into _____ different categories.	2	3	4	5	b
4	Which of the following is not consider laws for E-Type software evolution?	Continuing quality	Continuing change	Increasing complexity	Self-regulation	a
5	Which of the following laws for E-Type says "E-type software system must continue to adapt to the real world changes, else it becomes progressively less useful".	Continuing growth	Continuing change	Conservation of familiarity	Self-regulation	b
6	Which of the following is the Characteristics of good software?	Transitional	Operational	Maintenance	All of the above	d
7	Where there is a need of Software Engineering?	For Large Software	To reduce Cost	Software Quality Management	All of the above	d
8	The reason for software bugs and failures is due to_____.	Software Developers	Software companies	Both A and B	None of the above	c
9	Efficiency in a software product does not include _____	licensing	processing time	responsiveness	memory utilization	a
10	What are attributes of good software ?	Software functionality	Software development	Software maintainability	Both A and C	d

11	Different activity of a project management is	project planning	project monitoring	project control	All of the above	d
12	Which of the following activity is undertaken immediately after feasibility study and before the requirement analysis and specification phase?	Project Planning	Project Monitoring	Project Control	Project Scheduling	a
13	This activity is undertaken once the development activities start?	Project Planning	Project Monitoring and Control	Project size estimation	Project cost estimation	b
14	Which of the following activity is not the part of project planning?	project estimation	project scheduling	project monitoring	risk management	c
15	In the project planning, which of the following is considered as the most basic parameter based on which all other estimates are made?	project size	project effort	project duration	project schedule	a
16	During project estimation, project manager estimates following	project cost	project duration	project effort	all of the above	d
17	Once project planning is complete, project managers document their plan in	SPMP document	SRS document	Detailed Design documet	Excel Sheet	a
18	COCOMO stands for _____ .	COnsumed COst MOdel	COnstructive COst MOdel	COmmon COntrol	COmposition COst Model	b
19	Which may be estimated either in terms of KLOC (Kilo Line of Code) or by calculating number of function points in the software?	Time estimation	Effort estimation	Cost estimation	Software size estimation	d
20	Which metrics are derived by normalizing quality and/or productivity measures by considering the size of the software that has been produced?	Size oriented	Function-Oriented	Object-Oriented	Use-case-Oriented	a
21	What is the most common measure for correctness?	Defects per KLOC	Errors per KLOC	$ per KLOC	Pages of documentation	a
22	Abbreviate the term SMI.	Software Maturity Index	Software Model Instruction	Software Maturity	Software Model Index	a
23	Risk management is one of the most important jobs for a	Client	Investor	Production team	Project manager	d
24	Which of the following risk is the failure of a purchased component to perform as expected?	Product risk	Project risk	Business risk	Programming risk	a

67	Which of the following is a type of Architectural Mc ?	Static structural model	Dynamic process model	Distribution model	All of the mentione	d
68	Which of the following is golden rule for interface design?	Place the user in control	Reduce the user's memory load	Make the interface consistent	All of the mentione	d
69	Which of the following is not a design principle that allow the user to maintain control?	Provide for flexible interact	Allow user interaction to be interrupt-able and	Show technical internals from	Design for direct interaction with objects that appear	c
70	Which of the following is not a user interface design process?	User, task, and environment	Interface design	Knowledgeab le, frequent users	Interface validation	c
71	When users are involved in complex tasks, the demand on _____ can be significant.	short-term memory	shortcuts	objects that appear on the screen	all of the mentioned	a
72	Which of the following option is not considered by the Interface design?	the design of interfaces betv	the design of interfaces between the software and	the design of the interface between two	all of the mentioned	a
73	A software might allow a user to interact via	keyboard commands	mouse movement	voice recognition commands	all of the mentioned	d
74	What incorporates data, architectural, interface, and procedural representations of the software?	design model	user's model	mental image	system image	a
75	What establishes the profile of end-users of the system?	design model	user's model	mental image	system image	b
76	Deep structure testing is not designed to	examine object behaviors	exercise communic	exercise object	exercise structure o	d
77	What is the normal order of activities in which traditionalsoftware testing	integration testing, system t	unit testing, validat	unit testing, int	validation testing, s	c
78	Which of the following strategic issues needs to beaddressed in a successf	conduct formal technical re	specify requiremer	use independe	a and b	e
79	Bottom-up integration testing has as it's major advantage(s)that	major decision points are te	no drivers need to	no stubs need t	regression testing is	c
80	Smoke testing might best be described as	bulletproofing shrink-wrapp	rolling integration	testing that hid	unit testing for sma	b

S.NO	NAME OF THE STUDENT PLACED	ENROLMENT NO	NAME OF THE EMPLOYER	LETTER REFERENCE NO	SALARY
18	KOKILA R	621614104024	COINTONA IT TECHNOLOGIES INDIA PRIVATE LIMITED	CNT/18/007	120000
19	KOWSALYA B	621614104025	ACURUS SOLUTIONS PVT LTD	ARS/18/222	300000
20	KUNKALA VIJAY SAI PRANEETH	621614104026	INSTITUTE of LANGUAGE MANAGEMENT (P) Ltd	CI/2018-19/1090/A	192000
21	MADDI REDDY GARI RUPENDRA REDDY	621614104027	COINTONA IT TECHNOLOGIES INDIA PRIVATE LIMITED	CNT/18/008	120000
22	MAGESHWARI S	621614104028	MOBIUS KNOWLEDGE SERVICES	Mob/18-19/0105	165000
23	MENAKA R	621614104029	COINTONA IT TECHNOLOGIES INDIA PRIVATE LIMITED	CNT/18/009	120000
24	MOHANAPRIYA S	621614104030	COGNIZANT TECHNOLOGY SOLUTIONS	11898219	253158
25	MOHANAPRIYA S	621614104031	COINTONA IT TECHNOLOGIES INDIA PRIVATE LIMITED	CNT/18/010	120000
26	NELLORE SAIKRISHNA	621614104032	INSTITUTE of LANGUAGE MANAGEMENT (P) Ltd	CI/2018-19/1089/A	192000
27	NITHIYANANTHAM L	621614104033	INSTITUTE of LANGUAGE MANAGEMENT (P) Ltd	CI/2018-19/1087/B	192000
28	PAVITHRA G	621614104034	GODB TECH PRIVATE LIMITED	542	144000
29	PAVITHRA M	621614104035	INSTITUTE of LANGUAGE MANAGEMENT (P) Ltd	CI/2018-19/1096/A	192000
30	PAVITHRA P R	621614104036	INSTITUTE of LANGUAGE MANAGEMENT (P) Ltd	CI/2018-19/1099/B	192000
31	PRAGATHI KAVYAA N	621614104037	HEXAWARE TECHNOLOGIES	HW/18/255	240000
32	PRAKASH C	621614104038	COINTONA IT TECHNOLOGIES INDIA PRIVATE LIMITED	CNT/18/011	120000
33	PRASANTH P	621614104039	COINTONA IT TECHNOLOGIES INDIA PRIVATE LIMITED	CNT/18/012	120000

Estd. 2000

OFFICE OF THE CONTROLLER OF EXAMINATIONS

BE / B.Tech END SEMESTER EXAMINIATIONS

MAY / JUNE - 2021

BRANCH: COMPUTER SCIENCE AND ENGINEERING **SEMESTER: IV**

S.No	Reg. Number	Name of the Student	19CSC07	19CSC09	19CSC11	19CSC12	19CSC13	19CSE01	19CSC08	19CSC10	19CSE02
1	18CS117	VENMATHI G	B	B+	B+	B+	B	B+	A+	A+	A+
2	19CS001	AKASH KUMAR M R	A	B+	A	A	A	A	A+	A+	A+
3	19CS002	AMANULLA M	B	A	B+	B	B	B	B+	B+	B+
4	19CS004	ARUNKUMAR V	B	B+	B+	B+	B	B	A	A	A
5	19CS005	ASWIN RAJA K	B+	B+	B+	B	A	B+	A+	A	A+
6	19CS006	BAVADHARINI E	A+	A+	O	O	A+	A+	O	O	O
7	19CS007	BAVITHRAN K	B+	B	B	B+	B	B+	A	B+	A
8	19CS008	BHARATHI V	A	B	B+	A	B+	A	A+	A+	A+
9	19CS009	CHANDHRU G	B	B	B+	B	A	B+	A	A	A
10	19CS010	CYRILROY S	B+	B	B	B	B+	B+	A	A	A
11	19CS011	DANIEL SANTHOSH Y	B	B	B	B+	A	B	B+	B+	B+
12	19CS012	DEVADHARSHINI R	B+	B+	B+	A	B+	B+	A+	A+	A+
13	19CS013	DHANEESH M	B	B+	A	A	A	B+	A+	A+	A+
14	19CS014	DHARANESH G	B	B	B	B	B+	B	A	B+	B+
15	19CS015	ELAVARASAN S	B	B	B	B	B	A	A	B+	A
16	19CS016	FOLK N	A	B+	A	B+	A	A+	A+	A+	A+
17	19CS017	GOBIKRISHNAN R	B+	B	B	B+	B+	B+	A	A	A
18	19CS018	GOKULA KANNAN M	A	B+	A	A	A	A+	A	A+	A
19	19CS019	GOPI S	B	B	B	B	B	B	B+	B+	B+
20	19CS020	GOWTHAM R	B	B	B	B	B	B+	A	A	A
21	19CS021	GURUSARAN E	B	B	B	A	B+	B	A	A	A
22	19CS022	HARIPRASATH M	B+	B	B	B	B+	B	A	A	A
23	19CS023	HENDRY CHARLES G	B	B	A	B+	B+	B	A	A	A
24	19CS024	JAGADEESWARAN P	B+	B	B+	B+	B	B+	A+	A	A+
25	19CS025	JAGATH K R	B	B	A	B	A	B	B+	B+	B+
26	19CS026	JAWAHARBABU R	A	A	A	A	A	A	O	A+	O
27	19CS027	JAYASHANGAREE S	A+	A	O	A+	O	A+	A+	O	A+

S.No	Reg. Number	Name of the Student	19CSC07	19CSC09	19CSC11	19CSC1	19CSC13	19CSE01	19CSC08	19CSC10	19CSE02
28	19CS028	JAYASURYA C	B	B+	B+	B	A	B+	A	A	A
29	19CS029	JOSHINI M	A	B+	B	B+	B	B+	A	A	A
30	19CS030	KARTHIKEYAN K	B	B	B	A	B	A	B+	B+	B+
31	19CS031	KAVIN K R	B	A	B+	B	B	A	B+	A	B+
32	19CS032	KAVIN R	B	B	B+	B	B	B	B+	B+	B+
33	19CS033	KUMAR M	B	A	B	A	B	B	B+	B+	B+
34	19CS034	LOKESHWARAN V	B+	B+	B	B+	B+	B	A	A	A
35	19CS035	MANIKANDAN J	B	B	A	B+	B+	B+	A	A	A
36	19CS036	MOHANARAMAN V S	B+	B+	A	A	A	B+	A	A	A
37	19CS037	MUKESH KANNA R	B+	B	A	A	A	A	A	A	A
38	19CS038	MURALITHARAN G	B	B	B	A	B+	B	A	A	A
39	19CS039	NAVEEN S	B+	B+	A	A+	A	B	A+	A	A+
40	19CS040	PRADAP K	B+	B	B	B	B+	B+	A	A	A
41	19CS041	PRADEEPROSHAN R R	B	B	A	B	B+	B	A	B+	B+
42	19CS042	PRAVEEN P	B	B	B	B+	B+	A	B+	A	B+
43	19CS043	PRAVEEN S (15.S.2UU1)	B	B	B	B	B	A	B+	B+	B+
44	19CS044	PRAVEEN S (15.12.2UUU)	A+	A	A+	A+	A+	A	O	O	O
45	19CS045	PRIYA DHARSHINI S	A+	A	A+	O	A+	A	A+	A+	A+
46	19CS046	SARAN M	B	B	B+	A	B+	A	A	A	A
47	19CS047	SIVANESAN G	A	B+	A+	A	A+	A	A+	A+	A+
48	19CS048	SNEGA T	O	A	O	O	O	A+	O	O	O
49	19CS049	SRIDHAR C	B+	B	B+	A	B+	A	A	A	A
50	19CS050	SRIDHAR S	B	B	B	B	B	A	A	A	A
51	19CS051	SUDEEP P	B	B	B	B	B+	A	A	A	A
52	19CS052	SUDHARSAN R	B+	B	B	B+	B+	A	B+	A	B+
53	19CS053	SUREKA M	A	A	A	A	B+	A	A	A+	A
54	19CS054	SWATHI C	A	A	A	A	B+	B+	A+	A	A+
55	19CS055	VENKATESAN M	A	B	B+	B+	A	B+	A	A	A
56	19CS056	VIDHYASAGAR J	B	B+	B+	B+	A	B+	A	A+	A
57	19CS057	VIGNESH P	A+	A	A	A	A	A	A+	A+	A+
58	19CS058	VIHASINI M	B	B	B	B	A	B	B+	B+	B+

S.No	Reg. Number	Name of the Student	19CSC0	19CSC0	19CSC1	19CSC	19CSC1	19CSE01	19CSC0	19CSC1	19CSE0
59	19CS059	VIMALKUMAR M	B+	B	B	B	B	B+	A	A	A
60	19CS060	YUKENTHIRAN V	A	B+	A	A	A	A+	A+	A+	A+
61	19CS061	ATLA GAYATHRI	A+	A	A+	A+	A+	A+	A	A	A
62	19CS063	ATLA SRUTI	A+	O	A+	A+	A+	A+	A+	A+	A+
63	19CS064	BOMMEPALLE HARSHA VARDHAN REDDY	B+	B	A	B	A	B+	B+	A	B+
64	19CS065	BUCHI DHANUSH	A	A	A	A+	A	A+	A+	A+	A+
65	19CS066	C DINESH	A+	A+	A+	A	A+	A+	A+	A+	A+
66	19CS067	DANDE PAVAN KALYAN	A	A	A	A	B+	A	A	A	A
67	19CS068	EDDAGOTTI ANIL RAJ	B	B	B	A	B	A	B+	B+	B+
68	19CS069	GNANASHANKARAN N	A	B+	A	B+	A	A	A+	A	A+
69	19CS070	GOKULKUMAR M	A+	A+	A+	A+	A	A+	A+	A+	A+
70	19CS071	GUGGULLA HARISH REDDY	A	B+	A	A	A	A	A	A+	A
71	19CS072	K VAISHNAVI	A+	A+	A+	A+	A+	A+	A+	A+	A+
72	19CS073	KADURU ROHITH	A	A	A	B+	A	A	A	A	A
73	19CS074	KALAVAPALLI SATHISH REDDY	A	A	A	A	A	A+	A+	A	A+
74	19CS075	KARODI SURESH REDDY	A	B+	B+	A	A	A	A+	A	A+
75	19CS076	KAVIYARASAN P	A+	A	A+	A	A	A+	A+	A+	A+
76	19CS077	KODURU VISHNU VARDHAN REDDY	B+	A	A	A	A	B+	B+	A	B+
77	19CS078	KOMMERLA MAHIMA	A	A	A	A+	A	A+	A+	A+	A+
78	19CS079	KONDURU SOHITH VARMA	B	A	B	B	B	A	B+	B+	B+
79	19CS080	KOTIPALLI MANOJ KUMAR	B	B	B	B	B	A	B+	A	B+
80	19CS081	KOVURU VENKATASAI SUMANTH	A+	O	A+	A+	A+	A+	O	A+	O
81	19CS082	KUMMARA ARAVIND	B+	B+	A	A	B+	B+	A	A	A
82	19CS083	LAKSHMISETTY HEMANTH SAI	A+	A	A+	A+	A	A+	A+	A+	A+
83	19CS084	LINGI REDDY MAHANTH KUMAR REDDY	B	B	B	A	B	A	B+	A	B+
84	19CS085	M LOKESH	A+	A+	A+	A	A+	A+	O	A+	O
85	19CS086	MADALA VINAY	B	A	B	A	B	B	B+	B+	B+
86	19CS087	MADISETTY VEERA BHASKAR	B	A	B	B	B	A	B+	B+	B+
87	19CS088	MANGALA MUNIVARDHAN	B	B	B	B	A	A	B	B+	B

S.No	Reg. Number	Name of the Student	19CSC07	19CSC09	19CSC11	**19CSC1**	19CSC13	19CSE01	19CSC08	19CSC10	19CSE02
88	19CS089	MANJU V	A+	A+	A+	**A+**	A+	A+	A+	A+	A+
89	19CS090	MIDDISUNDAR BHARATH KUMAR	B	B	A	**B**	A	B	B+	A	B+
90	19CS091	MUKESH A	A	A	A	**A**	A+	A+	A+	A+	A+
91	19CS092	MUMMADI PRAVEEN KUMAR REDDY	B+	B+	A	**A**	B+	A	A+	A+	A+
92	19CS093	NAIDUGARI NARASIMHA NAIDU	A+	A	A+	**A**	A+	A+	O	O	O
93	19CS094	OBULIPURUSOTHAMAN K	A	A	A+	**A**	A	A+	A+	A+	A+
94	19CS095	PATNAM BABAFAKRUDDIN	B	B	A	**B**	B	B	A	B+	B+
95	19CS096	PEYYALA. PRUDHVI	A	A	A	**A**	A	A+	A+	A+	A+
96	19CS097	PONAGANI KUSUMASREE	A+	A+	A+	**A+**	A+	A+	A+	A+	A+
97	19CS098	PONNAM MANIKUMAR	B	B	A	**B**	A	B+	B+	B+	B+
98	19CS099	PONNAPATI DEEPAK KUMAR REDDY	B+	B+	B+	**B+**	A	B+	A	A	A
99	19CS100	PONNAPATI DINESH KUMAR REDDY	B+	B+	B+	**B+**	A	A	A	A	A
100	19CS101	POREDDY RANJITH KUMAR REDDY	B+	A	B+	**B+**	A	A	A+	A	A+
101	19CS102	POTHUPETA BHARGHAV	B+	A	B+	**B+**	B+	B+	A	A	A
102	19CS103	TELLA PRASANTH KUMAR	B+	B+	B+	**A**	B+	B+	A	A	A
103	19CS104	RACHURI LOKESH	B	B	B	**A**	B	B	A	B+	A
104	19CS105	RITHIKA V	A+	A+	A+	**A+**	A+	A+	O	A+	O
105	19CS106	RUDRAPATI VISHNUVARDHAN	B+	B+	B+	**A**	B+	A	A	B+	A
106	19CS107	SAMUKTHA M	A+	A	A+	**A+**	A	A+	A+	A+	A+
107	19CS108	SANDHI BOINA SRINIVASULU	A	B+	A	**A**	B+	A	A	A+	A
108	19CS109	SARAVANAN G	B+	B+	A	**B+**	B+	A	A	A	A
109	19CS110	SELVAKUMAR R	A	A+	A	**A**	A	A	A	A+	A
110	19CS111	SHANKAVI M	A+	A+	A+	**A+**	A+	A+	A+	A+	A+
111	19CS112	SIVAKUMAR S K	A+	A+	A+	**A+**	A	A+	A+	A+	A+
112	19CS113	SOWNDHAR S	A+	A+	A+	**A+**	A+	A+	O	A+	O
113	19CS114	TELAGARA VISHNUKANTH REDDY	B+	B+	A	**A**	B+	A	A	A	A
114	19CS115	THALISETTY KISHORE	A	A	A	**A**	B+	A	A+	A	A+
115	19CS116	VALLURU PREETHIKA	A+	A	A+	**A+**	A	A+	O	A+	O
116	19CS117	VANDHAVASI YUVARAJU	A	A+	A+	**A**	A	A+	A+	A+	A+

S.No	Reg. Number	Name of the Student	19CSC07	19CSC09	19CSC11	19CSC1	19CSC13	19CSE01	19CSC08	19CSC10	19CSE02
118	19CS118	VASANTHAPRIYA N J	A+	A+	A+	A+	A+	O	O	A+	O
119	19CS119	VELPULA JAYA KRISHNA YADAV	B+	B+	A	A	B+	A	A	A+	A
120	20CS501	AKASH A	B	B+	B+	A	A	B	A	A	A
121	20CS502	DHEVA M	A	B+	A	A+	A	A+	B+	A	B+
122	20CS503	DHIVYA R	B+	A	A	B+	A	A	A	A	A
123	20CS504	DINESH B	U*	U*	U*	U*	U*	U*	U*	U*	U*
124	20CS505	HEMANTH E C	A	A	A	A	O	A+	A	A+	A
125	20CS506	JAMUNA S	A	O	A+	A+	A+	A	A+	O	A+
126	20CS507	KEERTHANA T	U*	U*	U*	U*	U*	U*	U*	U*	U*
127	20CS508	MANIKANDAN M	U*	U*	U*	U*	U*	U*	U*	U*	U*
128	20CS509	MUTHUVEL A	B+	B	A	A	A+	A	A	A	A
129	20CS510	PRAVEENKUMAR V	U*	U*	U*	U*	U*	U*	U*	U*	U*
130	20CS511	RANJITH M	U*	U*	U*	U*	U*	U*	U*	U*	U*
131	20CS512	SOUNDHARYA S	A	A+	A+	A	A	B+	O	A+	O
132	20CS513	SOUNDHAR S	A	B	B+	B+	B+	A	B+	A+	B+
133	20CS514	VIMAL S	A	B+	A+	B+	A	B+	A+	A+	A+
134	20CS515	K.KIRITHISWARAN	B	B	B+	B+	A	B+	A+	A+	A+
135	20CS516	S.TAMILSELVAN	B	A	B+	A	B+	B+	A+	A+	A+

(An Autonomous Institution)
(Approved by AICTE, New Delhi, Accredited by NAAC & Affiliated to Anna University)
Rasipuram - 637 408, Namakkal Dist., Tamil Nadu.
Estd. 2000

Department of Computer Science and Engineering
Suggestion for the Improvement / Action Taken for the Course
Academic Year (2020-21)

Course Code & Course Name : 19CSC12 Software Engineering **Date : 25.4.2021**

- The target fixed for the Course in Corse Committee Meeting – I is 2
- The attained target at the end of the Course is

Course Outcome (CO)	Target
CO1	**2.15**
CO2	**2.00**
CO3	1.80
CO4	1.50
CO5	**2.15**

Points Discussed:

To attain the fixed target for the next batch, the Course Coordinator suggested the Course faculty to give focus on the following

- Providing more Hands on-training to the students on this Course
- Instruct the students to learn unit iii and unit iv in YouTube
- Provide the Lecture videos to the students for next semester

1. 25/4/21
2.

Course Faculty **Course Coordinator** **HOD**

Reg. No. : | | | | | | | |

MUTHAYAMMAL ENGINEERING COLLEGE (Autonomous), Rasipuram – 637 408
(Approved by AICTE & Affiliated to Anna University)

Question Paper Code: 194014

B.E./B.Tech. DEGREE EXAMINATIONS, APRIL/MAY 2019
Fourth Semester – Computer Science and Engineering/Information Technology
16CSD04/16ITD04 – OBJECT ORIENTED SOFTWARE ENGINEERING
(Regulations 2016)

Time: Three hours Maximum: 100 ma

Answer ALL questions

PART A – (10 × 2 = 20 Marks)

1. Define agile principle.
2. Differentiate between process and methodology.
3. List out the steps for requirement elicitation.
4. What is 3-board process?
5. What are the benefits of creator pattern?
6. What is actor system interaction modeling?
7. What are the tools supports for testing?
8. List out the various merits of test driven.
9. What is known as effort estimation method?
10. What is the CMMI?

PART B – (5 × 16 = 80 Marks)

11. a) i) Explain briefly about software lifecycle activities. (8
    ii) Write a short note on architectural design. (8

(OR)

    b) i). What are the software process models? Explain. (16)

12. a) Discuss in detail about the sequence diagram and class diagram with neat sketch. (16

(OR)

    b) Briefly explain about software requirement elicitation. (16)

13. a) What are the steps involved in driving a user interface design? Describe. (16)

(OR)

    b) i) Explain in detail about actor-system interaction modeling. (8)
    ii) Write short note on real time software design. (8)

14. a) Elaborate the conventional white-box testing. (16)

(OR)

    b) Explain about software quality assurance. (16)

15. a) Discuss in detail about: (i).Function point method (ii). COCOMO II Model. (16)

(OR)

    b) Describe in detail about Risk management. (16)

Reg. No. : | | | | | | | | | | | | |

Question Paper Code : 91399

B.E./B.Tech. DEGREE EXAMINATIONS, NOVEMBER/DECEMBER 2019
Fourth Semester
Computer Science and Engineering
CS 6403 – SOFTWARE ENGINEERING
(Common to Information Technology)
(Regulations 2013)
(Also common to PTCS 6403 – Software Engineering for B.E. (Part-Time) – Fourth Semester – Computer Science and Engineering – Regulations 2014)

Time : Three Hours | Maximum : 100 Marks

Answer ALL questions

PART – A **(10×2=20 Marks)**

1. Suggest a model to be used when enough staffing is unavailable and why.

2. State the pros and cons of COCOMO model.

3. Identify the notations for requirements specification.

4. State the applications of petri nets.

5. What is the use of fan in and fan out ?

6. Distinguish between class based components and traditional components.

7. How to calculate the reliability of the module ?

8. "Integration testing is harder than unit testing". Justify.

9. Estimate the function point for the below system.

Using the following table for function point weightings :

Factors	Weights		
	Simple	Average	Complex
Number of user inputs	3	4	6
Number of user outputs	4	5	7
Number of user inquiries	3	4	6
Numer of files	7	10	15
Number of external interfaces	5	7	10

A system being developed has the following characteristics :

Number of user inputs	10 (simple)
Number of user outputs	7 (simple)
Number of user inquiries	3 (average)
Number of files	6 (average)
Number of external interfaces	1 (complex)

10. Predict the expected cost for any branch of the decision tree in Make / Buy decision scenario.

PART – B **(5×13=65 Marks)**

11. a) i) Explain the term "Engineering" in Software Engineering. **(3)**

ii) Describe at least one scenario where 'RAD model would be applicable than not the waterfall model'. **(10)**

(OR)

b) i) Summarize in detail about risk management. **(5)**

ii) Elaborate on how LOC and FP can be used in project estimation. **(8)**

12. a) A software system is to be developed to automate a library catalogue. This system will contain information about all the books in a library and will be usable by library staff and by book borrowers and readers. The system should support catalogue browsing, querying, and should provide facilities allowing users to send messages to library staff reserving a book which is on loan.

For the above specification mention sketch the outline of requirements document as per the lEEE standard format.

(OR)

b) Illustrate in detail about

i) Petri nets **(6)**

ii) Data Dictionary. **(7)**

13. a) Outline clearly the concepts and types of coupling and cohesion with examples of each.

(OR)

b) Design and illustrate the user interface design for an webpage advertising underwater submarine.

14. a) Demonstrate the differences between black-box and structural testing and suggest how they can be used together in the defect testing process.

(OR)

b) i) Identify the purpose of regression testing. What are the two main activities of regression testing ? **(9)**

ii) Why do we need validation testing ? Explain. **(4)**

15. a) Explain in detail about the various phases, steps and activities that are needed for planning and managing a project with an illustration.

(OR)

b) Describe in detail about :

i) Risk Mitigation, Monitoring and Management Plan (RMMM) **(8)**

ii) Earned Value Analysis (EVA). **(5)**

PART – C **(1×15=15 Marks)**

16. a) Given,

Number of user inputs = 15

Number of user outputs = 13

Number of external interfaces = 11

1 function point = 20 LOC (as fourth generation language is used).

Values of constant used in basic COCOMO model. a = 2.4, b = 1.05, c = 2.5, d = 0.38.

Calculate and evaluate the effort and duration using the above details for basic COCOMO model.

(OR)

b) For each of the following types of projects, choose the most appropriate life cycle model and justify your choice by a couple of lines of explanation

i) You are migrating a legacy application in mainframes to Oracle. The project goes through well-defined phases of contract signing, taking each program of the current system with a well-defined acceptance test data, converting it to Oracle and proving that the output matches the expected output. It is not possible to seek intermediate feedback. **(8)**

ii) You are developing a proof-of-concept to show your prospect on how your product is suited for developing wireless applications. You do not have access to expensive CASE tools. **(7)**

Muthayammal Engineering College

CIA-I

Department of Computer Science and Engineering

Year/Sem: II/IV | R2016

Batch: 2019-2023 | Total Students

Name of the Faculty: Mr.S.R.Sridhar & Mr.V.Karuppuchamy | No. Students Attained: 102 | 93

Course Code & Name: 19CSC12 & Software Engineering | % of CO Attainment: 76.1 | 69.4

S.No	Roll No.	Name of Student	Q.No	1	2	3	4	5	6	7	8	Total	CO Attainment Level								
			CO No.	1	1	2	2	2	1	1	2		CO1	CO2	CO3	CO4	CO5	CO1	CO2	CO3	CO4
			Mark	2	2	2	2	2	10	15	15	50	29	21	0	0	0	3	2		
1	18CS117	VENMATHI G		2	2	2	2	2	10	15	14	49	100.0	95.2				Y	Y		
2	19CS001	AKASH KUMAR M R		2	2	2	2	2	10	15	15	50	100.0	100.0				Y	Y		
3	19CS002	AMANULLA M		2	2	1	1	1	5	8	4	24	82.8	38.1				Y	N		
4	19CS004	ARUNKUMAR V		2	2	2	0	2	10	15	15	48	100.0	90.5				Y	Y		
5	19CS005	ASWIN RAJA K		2	2	2	2	2	4	5	6	25	44.8	57.1				N	N		
6	19CS006	BAVADHARINI E		2	2	2	2	2	10	15	15	50	100.0	100.0				Y	Y		
7	19CS007	BAVITHRAN K		2	2	2	2	2	10	14	8	42	96.6	85.7				Y	Y		
8	19CS008	BHARATHI V		2	2	2	2	2	10	15	14	49	100.0	95.2				Y	Y		
9	19CS009	CHANDHRU G		1	1	0	1	1	9	10	10	33	72.4	67.6				Y	N		
10	19CS010	CYRILROY S		2	2	2	1	1	6	8	7	29	65.5	52.4				Y	N		
11	19CS011	DANIEL SANTHOSH Y		2	2	2	1	1	9	11	10	38	82.8	66.7				Y	Y		
12	19CS012	DEVADHARSHINI R		1	1	2	1	2	6	9	10	22	58.6	52.4				N	N		
13	19CS013	DHANEESH M		2	2	1	1	2	10	7	3	28	72.4	57.1				N	N		
14	19CS014	DHARANESH G		1	1	1	1	2	9	15	9	39	89.7	76.2				Y	Y		
15	19CS015	ELAVARASAN S		2	2	2	2	2	10	11	13	44	96.6	90.5				Y	Y		
16	19CS016	FOLK N		2	2	1	1	1	5	3	4	19	41.4	38.1				N	N		
17	19CS017	GOBIKRISHNAN R		2	2	2	2	1	10	8	4	21	41.4	42.9				N	N		
18	19CS018	GOKULA KANNAN M		2	2	2	2	2	10	15	14	49	100.0	95.2				Y	Y		
19	19CS019	GOPI S		1	1	2	2	1	8	12	13	41	79.3	85.7				Y	Y		
20	19CS020	GOWTHAM R		1	2	1	2	2	5	6	8	27	58.6	61.9				N	Y		
21	19CS021	GURUSARAN E		2	1	1	2	1	9	12	13	41	82.8	81.0				Y	Y		
22	19CS022	HARIPRASATH M		2	1	1	0	2	5	6	8	25	48.3	52.4				N	N		

23	19CS023	HENDRY CHARLES G		0	0	2	2	0	0	8	5	17	26.5	27.3					N	N		
24	19CS024	JAGADEESWARAN P		2	1	2	2	1	10	11	12	43	90.7	84.0					Y	Y		
25	19CS025	JAGATH K R		2	2	2	1	1	6	8	7	29	62.0	52.0					N	N		
26	19CS026	JAWAHARBABU R		1	1	1	2	2	9	14	12	43	86.2	85.7					Y	Y		
27	19CS027	JAYASHANGAREE S		2	1	1	2	2	9	15	13	46	91.0	90.5					Y	Y		
28	19CS028	JAYASURYA C		1	2	2	2	0	6	8	7	28	58.1	55.1					N	N		
29	19CS029	JOSHINI M		2	2	2	2	2	8	6	7	31	62.1	61.9					Y	Y		
30	19CS030	KARTHIKEYAN K		2	1	2	2	2	8	9	9	36	74.1	71.4					Y	Y		
31	19CS031	KAVIN K R		1	2	2	2	2	6	8	7	30	58.6	61.9					N	N		
32	19CS032	KAVIN R		2	2	2	1	2	9	8	9	30	65.2	66.7					N	Y		
33	19CS033	KUMAR M		1	1	1	2	2	10	15	15	47	93.1	95.2					Y	Y		
34	19CS034	LOKESHWARAN V		2	2	1	2	2	10	15	14	48	100.0	90.5					Y	Y		
35	19CS035	MANIKANDAN J		1	2	1	2	1	8	14	13	42	86.2	81.0					Y	Y		
36	19CS036	MOHANARAMAN V S		1	2	1	2	2	8	8	7	31	65.5	57.1					Y	N		
37	19CS037	MUKESH KANNA R		1	2	2	2	2	8	7	8	32	62.1	66.7					Y	Y		
38	19CS038	MURALITHARAN G		1	1	1	1	1	5	9	10	29	55.2	61.9					N	Y		
39	19CS039	NAVEEN S		1	2	1	1	2	10	15	15	47	96.6	90.5					Y	Y		
40	19CS040	PRADAP K		2	2	1	2	0	5	5	9	26	48.3	52.4					N	N		
41	19CS041	PRADEEPROSHAN R R		1	1	1	1	1	0	5	4	14	24.1	23.8					N	N		
42	19CS042	PRAVEEN P		2	2	2	2	2	8	6	8	32	62.1	66.7					Y	Y		
43	19CS043	PRAVEEN S (15.5.2001)		2	2	2	1	1	6	9	8	31	65.5	67.0					Y	N		
44	19CS044	PRAVEEN S (15.12.000)		2	2	2	2	2	10	15	15	50	100.0	100.0					Y	Y		
45	19CS045	PRIYA DHARSHINI S		1	1	2	2	2	10	15	15	48	93.1	100.0					Y	Y		
46	19CS046	SARAN M		2	2	0	2	2	10	15	14	47	100.0	85.7					Y	Y		
47	19CS047	SIVANESAN G		1	2	2	1	2	10	15	14	47	86.2	90.5					Y	Y		
48	19CS048	SNEGA T		2	2	2	1	2	10	15	15	49	100.0	95.2					Y	Y		
49	19CS049	SRIDHAR C		2	1	2	2	1	7	7	0	22	58.6	23.8					N	N		

50	19CS050	SRIDHAR S		2	2	0	0	2	8	9	25	83.3	80.6					N	N	
51	19CS051	SUDEEP P		2	2	2	1	2	9	10	28	93.3	90.3					Y	N	
52	19CS052	SUDHARSAN R		2	2	2	0	2	9	11	28	93.3	90.3					N	N	
53	19CS053	SUREKA M		2	2	0	2	2	7	12	29	96.6	93.5					Y	Y	
54	19CS054	SWATHI C		2	1	0	2	2	8	12	27	90.0	87.1					Y	Y	
55	19CS055	VENKATESAN M		2	2	2	1	2	10	14	47	96.6	90.5					Y	Y	
56	19CS056	VIDHYASAGAR J		2	2	2	2	2	8	9	31	93.3	87.1					N	Y	
57	19CS057	VIGNESH P		2	2	2	1	2	10	12	45	93.3	87.1					Y	Y	
58	19CS058	VIHASINI M		2	2	0	0	2	8	10	29	96.6	93.5					N	N	
59	19CS059	VIMALKUMAR M		2	2	2	1	2	10	15	49	100.0	96.8					Y	Y	
60	19CS060	YUKENTHIRAN V		2	2	2	1	2	10	15	48	100.0	90					Y	Y	
61	19CS061	ATLA GAYATHRI		2	2	2	2	2	10	15	50	100.0	100.0					Y	Y	
62	19CS063	ATLA SRUTI		2	2	2	0	0	8	8	30	69.0	87					Y	N	
63	19CS064	BOMMEPALLE HARSHA VARDHAN REDDY		2	2	2	1	2	9	10	42	79.3	80.6					Y	Y	
64	19CS065	BUCHI DHANUSH		2	2	2	2	2	10	14	15	49	96.6	100.0					Y	Y	
65	19CS066	C DINESH		2	2	2	2	2	10	10	12	45	93.1	85.7					Y	Y	
66	19CS067	DANDE PAVAN KALYAN		2	2	2	2	1	10	10	14	45	89.7	90					Y	Y	
67	19CS068	EDDAGOTTI ANIL RAJ		2	2	2	2	2	10	15	8	43	100.0	96.8					Y	Y	
68	19CS069	GNANASHANKARAN N		2	2	2	2	2	9	5	6	30	62.1	67.7					Y	N	
69	19CS070	GOKULKUMAR M		2	0	1	1	0	4	5	12	15	37.9	33.9					N	N	
70	19CS071	GUGGULLA HARISH REDDY		2	2	2	2	2	8	11	9	38	79.3	71					Y	Y	
71	19CS072	K VAISHNAVI		2	2	2	2	1	9	14	12	44	93.1	83.9					Y	Y	
72	19CS073	KADURU ROHITH		1	1	1	2	2	9	0	5	21	37.9	42					N	N	
73	19CS074	KALAVAPALLI SATHISH REDDY		2	0	1	2	1	9	9	10	34	69.0	66.7					Y	Y	
74	19CS075	KARODI SURESH REDDY		1	2	2	2	2	10	15	15	49	96.6	100.0					Y	Y	
75	19CS076	KAVIYARASAN P		2	2	2	2	2	8	13	12	43	86.2	83.9					Y	Y	
76	19CS077	KODURU VISHNU VARDHAN REDDY		2	2	2	2	2	7	10	10	37	72.4	76.2					Y	Y	

77	19CS078	KOMMERLA MAHIMA		2	2	2	0	2	8	12	14	42	80.8	84						
78	19CS079	KONDURU SOHITH VARMA		2	2	2	2	1	8	7	6	33	63.5	68.7						
79	19CS080	KOTIPALLI MANOJ KUMAR		2	2	2	2	1	9	12	14	44	84.6	89.5						
80	19CS081	KOVURU VENKATASAI SUMANTH		2	2	2	2	2	6	10	9	34	65.3	71.9						
81	19CS082	KUMMARA ARAVIND		2	1	2	2	2	6	9	10	33	63.1	71.4						
82	19CS083	LAKSHMISETTY HEMANTH SAI		1	2	2	1	1	9	12	9	37	71.1	80.0						
83	19CS084	LINGI REDDY MAHANTH KUMAR REDDY		1	2	2	2	2	10	15	12	46	88.4	85.7						
84	19CS085	M LOKESH		1	2	0	2	2	8	12	12	41	78.8	85.7						
85	19CS086	MADALA VINAY		2	1	2	2	2	6	8	10	33	63.5	71.4						
86	19CS087	MADISETTY VEERA BHASKAR		2	1	2	2	1	6	5	14	37	71.1	80						
87	19CS088	MANGALA MUNIVARDHAN		1	2	1	2	2	8	10	9	35	67.4	66.7						
88	19CS089	MANJU V		1	1	2	2	2	10	15	15	48	92.3	100.0						
89	19CS090	MIDDISUNDAR BHARATH KUMAR		2	2	1	1	1	7	10	8	32	67.4	62.1						
90	19CS091	MUKESH A		1	1	2	2	2	6	4	3	20	38.9	42.9						
91	19CS092	MUMMADI PRAVEEN KUMAR REDDY		2	1	2	2	2	7	0	9	25	34.5	57.1						
92	19CS093	NAIDUGARI NARASIMHA NAIDU		2	2	2	2	2	10	15	15	50	100.0	100.0						
93	19CS094	OBULIPURUSOTHAMAN K		2	1	0	1	0	10	14	15	43	91.9	76.2						
94	19CS095	PATNAM BABAFAKRUDDIN		1	2	1	1	1	6	8	9	29	58.6	57.1						
95	19CS096	PEYYALA. PRUDHVI		2	1	1	2	2	10	15	15	48	96.6	95.2						
96	19CS097	PONAGANI KUSUMASREE		1	1	2	2	2	10	14	15	47	89.7	100.0						
97	19CS098	PONNAM MANIKUMAR		2	1	1	0	2	0	8	9	32	69.0	53.1						
98	19CS099	PONNAPATI DEEPAK KUMAR REDDY		2	2	2	2	2	6	8	10	34	62.1	76.2						
99	19CS100	PONNAPATI DINESH KUMAR REDDY		2	2	2	2	1	5	9	10	33	62.1	71.4						
100	19CS101	POREDDY RANJITH KUMAR REDDY		1	2	1	1	1	7	9	11	33	65.5	66.7						
101	19CS102	POTHUPETA BHARGHAV		1	1	1	0	1	5	9	8	26	55.2	47.6						
102	19CS103	TELLA PRASANTH KUMAR		1	1	1	1	1	10	9	9	33	72.4	57.1						
103	19CS104	RACHURI LOKESH		1	2	2	2	2	10	15	15	49	96.6	100.0						

104	19CS105	RITHIKA V		2	2	2	2	2	10	15	15	50	100.0	100.0					Y	Y
105	19CS106	RUDRAPATI VISHNUVARDHAN		2	2	2	2	2	10	15	15	50	100.0	100.0					Y	Y
106	19CS107	SAMUKTHA M		2	2	2	2	2	10	15	15	50	100.0	100.0					Y	Y
107	19CS108	SANDHI BOINA SRINIVASULU		2	2	2	2	2	8	10	10	38	76.0	76.0					Y	Y
108	19CS109	SARAVANAN G		1	1	0	1	1	2	0	0	16	32.0	32.0					Y	N
109	19CS110	SELVAKUMAR R		2	2	2	2	2	10	15	15	50	100.0	100.0					Y	Y
110	19CS111	SHANKAVI M		2	2	2	1	2	9	14	10	42	84.0	84.0					Y	Y
111	19CS112	SIVAKUMAR S K		1	1	2	1	2	6	9	0	22	44.0	44.0					N	N
112	19CS113	SOWNDHAR S		2	2	1	1	2	5	7	8	28	56.0	56.0					N	N
113	19CS114	TELAGARA VISHNUKANTH REDDY		1	1	1	1	2	9	15	9	39	80.0	80.0					Y	Y
114	19CS115	THALISETTY KISHORE		2	2	2	2	2	10	1	15	41	82.0	90.5					Y	Y
115	19CS116	VALLURU PREETHIKA		2	2	1	2	2	10	14	4	37	96.6	42.9					Y	N
116	19CS117	VANDHAVASI YUVARAJU		2	2	2	2	1	0	8	4	21	42.0	42.9					N	N
117	19CS118	VASANTHAPRIYA N J		2	1	1	2	2	10	15	14	47	96.6	90.5					Y	Y
118	19CS119	VELPULA JAYA KRISHNA YADAV		1	1	2	2	1	10	15	15	47	93.4	95.2					Y	Y
119	20CS501	AKASH A		1	2	1	2	2	9	12	13	42	84.0	85.7					Y	Y
120	20CS502	DHEVA M		2	2	2	2	2	10	15	15	50	100.0	100.0					Y	Y
121	20CS503	DHIVYA R		1	1	1	1	1	2	9	2	18	36.0	38.5					N	N
122	20CS504	DINESH B		2	2	2	2	2	10	15	14	49	100.0	95.2					Y	Y
123	20CS505	HEMANTH E C		2	2	2	2	2	10	15	15	50	100.0	100.0					Y	Y
124	20CS506	JAMUNA S		2	2	2	2	2	10	15	15	50	100.0	100.0					Y	Y
125	20CS507	KEERTHANA T		2	2	2	0	2	10	15	15	48	100.0	90.5					Y	Y
126	20CS508	MANIKANDAN M		2	2	2	2	2	4	5	6	25	48.0	45.2					N	N
127	20CS509	MUTHUVEL A		2	2	2	2	2	10	15	14	49	100.0	95.2					Y	Y
128	20CS510	PRAVEENKUMAR V		2	2	2	2	2	10	14	8	42	96.6	66.7					Y	Y
129	20CS511	RANJITH M		2	2	2	2	2	10	15	14	49	100.0	95.2					Y	Y
130	20CS512	SOUNDHARYA S		1	1	0	1	1	9	8	10	33	72.4	57.1					Y	N

131	20CS513	SOUNDHAR S																						
132	20CS514	VIMAL S																						
133	20CS515	K.KIRITHISWARAN																						
134	20CS516	S.TAMILSELVAN																						

25/3/2021

Course Faculty 25/3/2021

HOD 25/3

Muthayammal Engineering College

Year/ Sem : II/IV | R2016 | CIA-II

Department of Computer Science and Engineering | Batch :2019-2023 | Total Students | 134

Name of the Faculty: Mr.S.R.Sridhar & Mr.V.Karuppuchamy | No. Students Attained | 85 | 87

Course Code & Name: 19CSC12 & Software Engineering | % of CO Attainment | 63.4 | 64.9

S.No	Roll No.	Name of Student		Q No.	1	2	3	4	5	6	7	8	Total	CO Attainment Level									
				CO Nos	CO5	CO5	CO1	CO1	CO3	CO4	CO4	CO4		CO1	CO2	CO2	CO3	CO3	CO4	CO5	CO1	CO2	CO5
				Marks	2	2	2	2	2	10	15	15	50	0	0	12	31	1	60	60	60	60	60
1	18CS117	VENMATHI G			2	2	2	2	2	10	15	14	49			100.0	96.8					Y	Y
2	19CS001	AKASH KUMAR M R			1	2	1	2	1	9	14	12	42			90.5	80.6					Y	Y
3	19CS002	AMANULLA M			2	1	2	1	2	5	8	7	28			52.6	54.8					N	Y
4	19CS004	ARUNKUMAR V			2	2	2	2	2	10	15	15	49			94.7	100.0					Y	Y
5	19CS005	ASWIN RAJA K			1	2	2	2	2	7	5	4	25			42.1	54.8					N	N
6	19CS006	BAVADHARINI E			2	2	2	2	2	10	15	14	49			100.0	96.8					Y	Y
7	19CS007	BAVITHRAN K			2	1	0	2	0	3	4	3	15			30.8	25.8					N	N
8	19CS008	BHARATHI V			2	2	2	2	2	10	15	15	50			100.0	100.0					Y	Y
9	19CS009	CHANDHRU G			1	1	1	1	1	7	10	12	34			68.4	71.0					Y	Y
10	19CS010	CYRILROY S			2	2	2	1	1	6	5	7	29			63.2	54.8					Y	N
11	19CS011	DANIEL SANTHOSH Y			1	1	2	1	2	10	7	16	39			78.9	90.3					N	Y
12	19CS012	DEVADHARSHINI R			1	1	2	1	2	0	14	15	36			84.2	64.5					Y	N
13	19CS013	DHANEESH M			2	2	1	1	2	5	10	8	32			78.9	54.8					Y	N
14	19CS014	DHARANESH G			1	1	1	1	0	5	5	10	26			47.4	54.8					N	N
15	19CS015	ELAVARASAN S			2	2	2	2	2	9	11	10	38			78.9	74.2					Y	Y
16	19CS016	FOLK N			2	2	1	1	1	5	0	6	18			21.1	45.2					N	N
17	19CS017	GOBIKRISHNAN R			2	2	2	2	1	5	0	5	19			21.1	38.7					N	N
18	19CS018	GOKULA KANNAN M			2	2	2	2	2	10	14	15	49			94.7	100.0					Y	Y
19	19CS019	GOPI S			1	1	2	2	1	4	9	0	20			57.9	29.0					N	N
20	19CS020	GOWTHAM R			1	2	1	2	2	5	6	8	27			47.4	58.1					N	N
21	19CS021	GURUSARAN E			2	1	1	2	1	8	10	10	35			68.4	71.0					Y	Y
22	19CS022	HARIPRASATH M			2	2	2	2	2	8	14	12	44			94.7	83.9					Y	Y

23	19CS023	HENDRY CHARLES G		0	0	0	0	0	0	0	0									N	N	
24	19CS024	JAGADEESWARAN P		2	1	2	2	1	7	10	12	40								Y	Y	
25	19CS025	JAGATH K R		2	2	2	2	1	6	8	6	29								N	N	
26	19CS026	JAWAHARBABU R		1	1	1	2	2	5	8	7	27								N	N	
27	19CS027	JAYASHANGAREE S		2	1	1	2	2	10	14	14	46								Y	Y	
28	19CS028	JAYASURYA C		1	2	0	2	2	8	9	8	34								N	N	
29	19CS029	JOSHINI M		2	2	2	1	2	9	8	7	33								Y	Y	
30	19CS030	KARTHIKEYAN K		2	2	2	2	2	8	14	13	45								Y	Y	
31	19CS031	KAVIN K R		1	0	2	2	0	5	8	9	27								N	N	
32	19CS032	KAVIN R		2	2	2	1	0	8	9	8	31								N	N	
33	19CS033	KUMAR M		1	1	1	2	2	10	15	15	47								Y	Y	
34	19CS034	LOKESHWARAN V		2	2	1	0	0	10	15	14	44								Y	Y	
35	19CS035	MANIKANDAN J		1	2	1	2	1	8	10	13	38								Y	Y	
36	19CS036	MOHANARAMAN V S		1	2	1	2	2	8	7	5	28								N	N	
37	19CS037	MUKESH KANNA R		1	2	2	2	2	8	9	7	33								Y	Y	
38	19CS038	MURALITHARAN G		1	1	1	1	1	5	9	10	29								N	N	
39	19CS039	NAVEEN S		1	2	1	2	2	4	0	3	15								N	N	
40	19CS040	PRADAP K		2	2	2	2	2	8	10	10	38								Y	Y	
41	19CS041	PRADEEPROSHAN R R		1	2	2	2	2	4	0	2	15								N	N	
42	19CS042	PRAVEEN P		2	2	1	1	0	8	9	8	32								Y	Y	
43	19CS043	PRAVEEN S (15.S.2001)		2	1	0	2	1	9	8	5	28								N	N	
44	19CS044	PRAVEEN S (15.12.000)		2	2	2	2	2	10	12	14	46								Y	Y	
45	19CS045	PRIYA DHARSHINI S		2	2	2	2	2	10	15	15	50								Y	Y	
46	19CS046	SARAN M		2	2	2	2	2	10	15	15	50								Y	Y	
47	19CS047	SIVANESAN G		1	2	2	1	2	10	15	14	47								Y	Y	
48	19CS048	SNEGA T		2	2	2	1	2	10	15	15	49								Y	Y	
49	19CS049	SRIDHAR C		2	2	2	2	2	9	14	8	41								Y	Y	

50	19CS050	SRIDHAR S		2	2	2	2	2	10	15	15	50			100.0	100.0				Y	Y
51	19CS051	SUDEEP P		1	2	2	2	2	7	10	10	34			68.4	67.7				Y	Y
52	19CS052	SUDHARSAN R		2	2	1	2	2	5	5	8	27			47.4	58.1				N	N
53	19CS053	SUREKA M		1	1	2	2	1	5	7	9	27			47.4	58.1				N	N
54	19CS054	SWATHI C		1	2	2	2	2	7	7	7	29			52.6	61.3				N	N
55	19CS055	VENKATESAN M		2	2	2	2	2	10	15	15	48			100.0	96.8				Y	Y
56	19CS056	VIDHYASAGAR J		1	2	2	2	2	6	8	9	30			57.9	61.3				N	Y
57	19CS057	VIGNESH P		2	2	2	2	2	10	15	14	47			94.7	93.6				Y	Y
58	19CS058	VIHASINI M		2	2	2	2	2	5	6	8	26			52.6	51.6				N	N
59	19CS059	VIMALKUMAR M		2	2	2	2	2	10	15	14	48			100.0	93.5				Y	Y
60	19CS060	YUKENTHIRAN V		2	2	2	2	2	10	15	14	48			100.0	93.5				Y	Y
61	19CS061	ATLA GAYATHRI		2	2	2	2	2	10	15	14	49			100.0	96.8				Y	Y
62	19CS063	ATLA SRUTI		1	2	2	2	1	9	10	17	34			68.4	67.7				Y	Y
63	19CS064	BOMMEPALLE HARSHA VARDHAN REDDY		2	2	2	1	2	9	10	10	38			73.7	77.4				Y	Y
64	19CS065	BUCHI DHANUSH		2	2	2	2	2	10	15	15	50			100.0	100.0				Y	Y
65	19CS066	C DINESH		2	2	1	1	2	7	10	10	34			73.7	64.5				Y	Y
66	19CS067	DANDE PAVAN KALYAN		2	2	2	2	2	9	11	12	42			78.9	87.1				Y	Y
67	19CS068	EDDAGOTTI ANIL RAJ		2	1	2	2	1	9	8	6	33			57.9	71.0				N	Y
68	19CS069	GNANASHANKARAN N		2	2	1	1	2	4	5	6	26			47.4	54.8				N	N
69	19CS070	GOKULKUMAR M		2	2	2	0	0	6	7	5	22			57.9	85.5				N	N
70	19CS071	GUGGULLA HARISH REDDY		2	2	2	2	1	8	9	9	35			68.4	77.0				Y	Y
71	19CS072	K VAISHNAVI		1	1	1	2	2	7	10	9	33			63.2	67.7				Y	Y
72	19CS073	KADURU ROHITH		2	0	1	2	1	5	2	0	13			21.1	29.0				N	N
73	19CS074	KALAVAPALLI SATHISH REDDY		1	1	1	2	2	7	7	7	28			47.4	61.3				N	Y
74	19CS075	KARODI SURESH REDDY		1	0	2	1	2	10	15	15	46			84.2	96.8				Y	Y
75	19CS076	KAVIYARASAN P		2	2	2	2	2	7	10	10	37			73.7	74.2				Y	Y
76	19CS077	KODURU VISHNU VARDHAN REDDY		1	1	1	2	2	10	15	15	48			89.5	100.0				Y	Y

77	19CS078	KOMMERLA MAHIMA		2	2	2	1	1	6	7	8	33			67.1	71.0				N	N	
78	19CS079	KONDURU SOHITH VARMA		2	2	2	2	1	6	7	5	27			47.4	51.6				N	N	
79	19CS080	KOTIPALLI MANOJ KUMAR		2	2	2	2	2	6	9	9	34			68.4	67.7				N	Y	
80	19CS081	KOVURU VENKATASAI SUMANTH		2	1	2	2	1	8	9	10	35			73.7	67.7				Y	Y	
81	19CS082	KUMMARA ARAVIND		1	2	2	2	1	9	12	12	40			78.9	80.6				Y	Y	
82	19CS083	LAKSHMISETTY HEMANTH SAI		1	1	1	1	1	5	10	7	28			47.4	54.8				N	N	
83	19CS084	LINGI REDDY MAHANTH KUMAR REDDY		1	2	1	2	2	8	9	10	32			63.2	64.5				Y	Y	
84	19CS085	M LOKESH		2	1	2	2	1	6	9	10	33			63.2	67.7				Y	Y	
85	19CS086	MADALA VINAY		2	1	2	2	1	6	9	6	29			63.2	54.8				Y	N	
86	19CS087	MADISETTY VEERA BHASKAR		0	0	0	0	0	0	0	0	0			0.0	0.0				N	N	
87	19CS088	MANGALA MUNIVARDHAN		1	2	2	2	2	8	10	9	35			68.4	71.0				Y	Y	
88	19CS089	MANJU V		2	2	2	2	0	6	13	13	39			89.5	71.0				Y	Y	
89	19CS090	MIDDISUNDAR BHARATH KUMAR		2	2	1	1	1	7	10	8	32			73.7	58.1				Y	N	
90	19CS091	MUKESH A		1	1	2	2	2	5	0	4	17			10.5	48.4				N	N	
91	19CS092	MUMMADI PRAVEEN KUMAR REDDY		2	1	2	2	2	5	7	7	28			52.6	58.1				N	N	
92	19CS093	NAIDUGARI NARASIMHA NAIDU		2	2	2	2	2	10	14	15	49			94.7	100.0				Y	Y	
93	19CS094	OBULIPURUSOTHAMAN K		2	1	2	1	2	6	5	6	24			42.1	51.6				N	N	
94	19CS095	PATNAM BABAFAKRUDDIN		1	2	1	1	1	6	8	9	29			57.9	58.1				N	N	
95	19CS096	PEYYALA. PRUDHVI		2	2	0	2	2	10	15	15	48			100.0	93.5				Y	Y	
96	19CS097	PONAGANI KUSUMASREE		2	2	2	2	2	10	14	14	47			89.5	96.8				Y	Y	
97	19CS098	PONNAM MANIKUMAR		2	1	1	0	2	5	7	7	25			52.6	54.8				N	N	
98	19CS099	PONNAPATI DEEPAK KUMAR REDDY		2	2	2	2	2	6	8	10	34			63.2	71.0				Y	Y	
99	19CS100	PONNAPATI DINESH KUMAR REDDY		2	2	2	2	1	5	9	10	33			68.4	64.5				Y	Y	
100	19CS101	POREDDY RANJITH KUMAR REDDY		0	2	1	1	1	7	9	11	32			57.9	67.7				N	Y	
101	19CS102	POTHUPETA BHARGHAV		1	2	1	0	1	6	7	8	26			52.6	51.6				N	N	
102	19CS103	TELLA PRASANTH KUMAR		1	1	2	1	1	7	9	10	32			57.9	62.7				N	Y	
103	19CS104	RACHURI LOKESH		2	2	2	2	2	10	15	15	50			100.0	100.0				Y	Y	

104	19CS105	RITHIKA V		1	2	2	2	2	7	12	11	39								Y	Y
105	19CS106	RUDRAPATI VISHNUVARDHAN		2	2	2	2	2	5	5	5	25								N	N
106	19CS107	SAMUKTHA M		2	0	0	2	0	10	15	12	41								Y	Y
107	19CS108	SANDHI BOINA SRINIVASULU		2	0	2	1	2	5	10	10	31								Y	Y
108	19CS109	SARAVANAN G		2	2	2	2	2	10	10	0	30								N	N
109	19CS110	SELVAKUMAR R		2	2	2	1	1	6	11	10	35								Y	Y
110	19CS111	SHANKAVI M		1	1	2	1	2	10	17	15	39								Y	Y
111	19CS112	SIVAKUMAR S K		1	1	2	1	2	0	11	15	36								Y	Y
112	19CS113	SOWNDHAR S		2	2	2	2	2	10	12	12	46								Y	Y
113	19CS114	TELAGARA VISHNUKANTH REDDY		2	2	1	2	2	5	10	10	37								Y	Y
114	19CS115	THALISETTY KISHORE		2	2	2	2	2	7	11	10	38								Y	Y
115	19CS116	VALLURU PREETHIKA		2	2	1	1	1	5	0	6	18								N	N
116	19CS117	VANDHAVASI YUVARAJU		2	2	2	2	1	10	15	5	39								Y	Y
117	19CS118	VASANTHAPRIYA N J		2	1	1	2	2	10	15	15	48								Y	Y
118	19CS119	VELPULA JAYA KRISHNA YADAV		1	1	2	2	1	10	15	14	46								Y	Y
119	20CS501	AKASH A		1	2	1	2	2	7	12	12	39								Y	Y
120	20CS502	DHEVA M		2	1	2	2	2	10	15	15	49								Y	Y
121	20CS503	DHIVYA R		1	1	1	1	1	2	7	5	19								N	N
122	20CS504	DINESH B		2	2	2	2	2	10	15	14	49								Y	Y
123	20CS505	HEMANTH E C		1	2	1	0	1	9	14	14	42								Y	Y
124	20CS506	JAMUNA S		2	2	2	2	2	10	15	15	50								Y	Y
125	20CS507	KEERTHANA T		2	1	2	2	2	10	15	15	49								Y	Y
126	20CS508	MANIKANDAN M		1	2	2	2	2	7	5	4	25								N	N
127	20CS509	MUTHUVEL A		1	1	0	1	0	5	5	5	18								N	N
128	20CS510	PRAVEENKUMAR V		2	1	0	2	0	3	4	3	15								N	N
129	20CS511	RANJITH M		2	2	2	2	2	10	15	15	50								Y	Y
130	20CS512	SOUNDHARYA S		1	1	1	1	1	7	10	12	34								Y	Y

131	20CS513	SOUNDHAR S		[redacted]																			
132	20CS514	VIMAL S		[redacted]																			
133	20CS515	K.KIRITHISWARAN		[redacted]																			
134	20CS516	S.TAMILSELVAN		[redacted]																			

28/4/2021

Course Faculty 28/4/2021

HOD 28/4

Muthayammal Engineering College					Year/ Sem : II/IV					R2016				Assignment I
Department of Computer Science and Engineering					Batch : 2019-2023					Total Students				13
Name of the Faculty: Mr.S.R.Sridhar & Mr.V.Karuppuchamy					No. Students Attained									96
Course Code & Name:19CSC12 & Software Engineering					% of CO Attainment									71.6
			Q.No.		CO Attainment Level									2
S.No	Roll No.	Name of Student	CO. No.	Total	CO1	CO2	CO3	CO4	CO5	CO1	CO2	CO3	CO4	CO5
			Marks	50	0	0	0	0	50	60	60	60	60	60
1	18CS117	VENMATHI G		50					100.0					Y
2	19CS001	AKASH KUMAR M R		41					82.0					Y
3	19CS002	AMANULLA M		28					56.0					N
4	19CS004	ARUNKUMAR V		41					82.0					Y
5	19CS005	ASWIN RAJA K		40					80.0					Y
6	19CS006	BAVADHARINI E		49					98.0					Y
7	19CS007	BAVITHRAN K		49					98.0					Y
8	19CS008	BHARATHI V		49					98.0					Y
9	19CS009	CHANDHRU G		34					68.0					Y
10	19CS010	CYRILROY S		41					82.0					Y
11	19CS011	DANIEL SANTHOSH Y		28					56.0					N
12	19CS012	DEVADHARSHINI R		33					66.0					Y
13	19CS013	DHANEESH M		32					64.0					Y
14	19CS014	DHARANESH G		26					52.0					N
15	19CS015	ELAVARASAN S		34					68.0					Y

16	19CS016	FOLK N		32					64.0			Y
17	19CS017	GOBIKRISHNAN R		24					48.0			N
18	19CS018	GOKULA KANNAN M		49					98.0			Y
19	19CS019	GOPI S		31					62.0			Y
20	19CS020	GOWTHAM R		39					78.0			Y
21	19CS021	GURUSARAN E		35					70.0			Y
22	19CS022	HARIPRASATH M		30					60.0			Y
23	19CS023	HENDRY CHARLES G		24					48.0			N
24	19CS024	JAGADEESWARAN P		28					56.0			N
25	19CS025	JAGATH K R		17					34.0			N
26	19CS026	JAWAHARBABU R		27					54.0			N
27	19CS027	JAYASHANGAREE S		34					68.0			Y
28	19CS028	JAYASURYA C		49					98.0			Y
29	19CS029	JOSHINI M		50					100.0			Y
30	19CS030	KARTHIKEYAN K		50					100.0			Y
31	19CS031	KAVIN K R		31					62.0			Y
32	19CS032	KAVIN R		29					58.0			N
33	19CS033	KUMAR M		34					68.0			Y
34	19CS034	LOKESHWARAN V		36					72.0			Y
35	19CS035	MANIKANDAN J		38					76.0			Y
36	19CS036	MOHANARAMAN V S		34					68.0			Y

37	19CS037	MUKESH KANNA R		29					58.0			N
38	19CS038	MURALITHARAN G		29					58.0			N
39	19CS039	NAVEEN S		28					56.0			N
40	19CS040	PRADAP K		30					60.0			Y
41	19CS041	PRADEEPROSHAN R R		26					52.0			N
42	19CS042	PRAVEEN P		31					62.0			Y
43	19CS043	PRAVEEN S (15.S.2001)		27					54.0			N
44	19CS044	PRAVEEN S (15.12.000)		50					100.0			Y
45	19CS045	PRIYA DHARSHINI S		50					100.0			Y
46	19CS046	SARAN M		50					100.0			Y
47	19CS047	SIVANESAN G		46					92.0			Y
48	19CS048	SNEGA T		49					98.0			Y
49	19CS049	SRIDHAR C		31					62.0			Y
50	19CS050	SRIDHAR S		20					40.0			N
51	19CS051	SUDEEP P		33					66.0			Y
52	19CS052	SUDHARSAN R		23					46.0			N
53	19CS053	SUREKA M		27					54.0			N
54	19CS054	SWATHI C		29					58.0			N
55	19CS055	VENKATESAN M		34					68.0			Y
56	19CS056	VIDHYASAGAR J		32					64.0			Y
57	19CS057	VIGNESH P		47					94.0			Y

58	19CS058	VIHASINI M		28					56.0				N
59	19CS059	VIMALKUMAR M		29					58.0				N
60	19CS060	YUKENTHIRAN V		49					98.0				Y
61	19CS061	ATLA GAYATHRI		49					98.0				Y
62	19CS063	ATLA SRUTI		48					96.0				Y
63	19CS064	BOMMEPALLE HARSHA VARDHAN REDDY		38					76.0				Y
64	19CS065	BUCHI DHANUSH		46					92.0				Y
65	19CS066	C DINESH		35					70.0				Y
66	19CS067	DANDE PAVAN KALYAN		28					56.0				N
67	19CS068	EDDAGOTTI ANIL RAJ		35					70.0				Y
68	19CS069	GNANASHANKARAN N		34					68.0				Y
69	19CS070	GOKULKUMAR M		14					28.0				N
70	19CS071	GUGGULLA HARISH REDDY		28					56.0				N
71	19CS072	K VAISHNAVI		35					70.0				Y
72	19CS073	KADURU ROHITH		33					66.0				Y
73	19CS074	KALAVAPALLI SATHISH REDDY		32					64.0				Y
74	19CS075	KARODI SURESH REDDY		49					98.0				Y
75	19CS076	KAVIYARASAN P		32					64.0				Y
76	19CS077	KODURU VISHNU VARDHAN REDDY		37					74.0				Y
77	19CS078	KOMMERLA MAHIMA		29					58.0				N
78	19CS079	KONDURU SOHITH VARMA		33					66.0				Y

79	19CS080	KOTIPALLI MANOJ KUMAR		27					54.0				N
80	19CS081	KOVURU VENKATASAI SUMANTH		34					68.0				Y
81	19CS082	KUMMARA ARAVIND		33					66.0				Y
82	19CS083	LAKSHMISETTY HEMANTH SAI		40					80.0				Y
83	19CS084	LINGI REDDY MAHANTH KUMAR REDDY		49					98.0				Y
84	19CS085	M LOKESH		32					64.0				Y
85	19CS086	MADALA VINAY		33					66.0				Y
86	19CS087	MADISETTY VEERA BHASKAR		29					58.0				N
87	19CS088	MANGALA MUNIVARDHAN		35					70.0				Y
88	19CS089	MANJU V		45					90.0				Y
89	19CS090	MIDDISUNDAR BHARATH KUMAR		32					64.0				Y
90	19CS091	MUKESH A		31					62.0				Y
91	19CS092	MUMMADI PRAVEEN KUMAR REDDY		28					56.0				N
92	19CS093	NAIDUGARI NARASIMHA NAIDU		49					98.0				Y
93	19CS094	OBULIPURUSOTHAMAN K		28					56.0				N
94	19CS095	PATNAM BABAFAKRUDDIN		29					58.0				N
95	19CS096	PEYYALA. PRUDHVI		19					38.0				N
96	19CS097	PONAGANI KUSUMASREE		48					96.0				Y
97	19CS098	PONNAM MANIKUMAR		27					54.0				N
98	19CS099	PONNAPATI DEEPAK KUMAR REDDY		34					68.0				Y
99	19CS100	PONNAPATI DINESH KUMAR REDDY		33					66.0				Y

100	19CS101	POREDDY RANJITH KUMAR REDDY		33					66.0				Y
101	19CS102	POTHUPETA BHARGHAV		30					60.0				Y
102	19CS103	TELLA PRASANTH KUMAR		32					64.0				Y
103	19CS104	RACHURI LOKESH		50					100.0				Y
104	19CS105	RITHIKA V		40					80.0				Y
105	19CS106	RUDRAPATI VISHNUVARDHAN		18					36.0				N
106	19CS107	SAMUKTHA M		14					28.0				N
107	19CS108	SANDHI BOINA SRINIVASULU		31					62.0				Y
108	19CS109	SARAVANAN G		34					68.0				Y
109	19CS110	SELVAKUMAR R		34					68.0				Y
110	19CS111	SHANKAVI M		28					56.0				N
111	19CS112	SIVAKUMAR S K		47					94.0				Y
112	19CS113	SOWNDHAR S		32					64.0				Y
113	19CS114	TELAGARA VISHNUKANTH REDDY		26					52.0				N
114	19CS115	THALISETTY KISHORE		38					76.0				Y
115	19CS116	VALLURU PREETHIKA		48					96.0				Y
116	19CS117	VANDHAVASI YUVARAJU		24					48.0				N
117	19CS118	VASANTHAPRIYA N J		48					96.0				Y
118	19CS119	VELPULA JAYA KRISHNA YADAV		47					94.0				Y
119	20CS501	AKASH A		30					60.0				Y
120	20CS502	DHEVA M		35					70.0				Y

121	20CS503	DHIVYA R		19					38.0				N
122	20CS504	DINESH B		50					100.0				Y
123	20CS505	HEMANTH E C		41					82.0				Y
124	20CS506	JAMUNA S		49					98.0				Y
125	20CS507	KEERTHANA T		41					82.0				Y
126	20CS508	MANIKANDAN M		40					80.0				Y
127	20CS509	MUTHUVEL A		49					98.0				Y
128	20CS510	PRAVEENKUMAR V		49					98.0				Y
129	20CS511	RANJITH M		49					98.0				Y
130	20CS512	SOUNDHARYA S		34					68.0				Y
131	20CS513	SOUNDHAR S		41					82.0				Y
132	20CS514	VIMAL S		28					56.0				N
133	20CS515	K.KIRITHISWARAN		33					66.0				Y
134	20CS516	S.TAMILSELVAN		32					64.0				Y

1. 3/5/2021

Course Faculty 3/5/2021

HOD 3/5

Muthayammal Engineering College					Year/ Sem : II/IV					R2016				Assignment II
Department of Computer Science and Engineering					Batch : 2019-2023					Total Students				1
Name of the Faculty: Mr.S.R.Sridhar & Mr.V.Karuppuchamy					No. Students Attained									98
Course Code & Name: 19CSC12 & Software Engineering					% of CO Attainment									73.1
S.No	Roll No.	Name of Student	CO No.	Total	CO Attainment Level									2
			CO No.		CO1	CO2	CO3	CO4	CO5	CO1	CO2	CO3	CO4	CO5
			Marks	50	0	0	0	0	50	60	60	60	60	60
1	18CS117	VENMATHI G		34					68.0					Y
2	19CS001	AKASH KUMAR M R		37					74.0					Y
3	19CS002	AMANULLA M		49					98.0					Y
4	19CS004	ARUNKUMAR V		47					94.0					Y
5	19CS005	ASWIN RAJA K		40					80.0					Y
6	19CS006	BAVADHARINI E		50					100.0					Y
7	19CS007	BAVITHRAN K		14					28.0					N
8	19CS008	BHARATHI V		50					100.0					Y
9	19CS009	CHANDHRU G		34					68.0					Y
10	19CS010	CYRILROY S		41					82.0					Y
11	19CS011	DANIEL SANTHOSH Y		28					56.0					N
12	19CS012	DEVADHARSHINI R		33					66.0					Y
13	19CS013	DHANEESH M		32					64.0					Y
14	19CS014	DHARANESH G		32					64.0					Y
15	19CS015	ELAVARASAN S		38					76.0					Y

16	19CS016	FOLK N		32					64.0				Y
17	19CS017	GOBIKRISHNAN R		24					48.0				N
18	19CS018	GOKULA KANNAN M		50					100.0				Y
19	19CS019	GOPI S		31					62.0				Y
20	19CS020	GOWTHAM R		39					78.0				Y
21	19CS021	GURUSARAN E		35					70.0				Y
22	19CS022	HARIPRASATH M		32					64.0				Y
23	19CS023	HENDRY CHARLES G		24					48.0				N
24	19CS024	JAGADEESWARAN P		32					64.0				Y
25	19CS025	JAGATH K R		17					34.0				N
26	19CS026	JAWAHARBABU R		27					54.0				N
27	19CS027	JAYASHANGAREE S		46					92.0				Y
28	19CS028	JAYASURYA C		35					70.0				Y
29	19CS029	JOSHINI M		38					76.0				Y
30	19CS030	KARTHIKEYAN K		50					100.0				Y
31	19CS031	KAVIN K R		22					44.0				N
32	19CS032	KAVIN R		29					58.0				N
33	19CS033	KUMAR M		34					68.0				Y
34	19CS034	LOKESHWARAN V		36					72.0				Y
35	19CS035	MANIKANDAN J		38					76.0				Y
36	19CS036	MOHANARAMAN V S		33					66.0				Y

37	19CS037	MUKESH KANNA R		30					60.0					Y
38	19CS038	MURALITHARAN G		50					100.0					Y
39	19CS039	NAVEEN S		28					56.0					N
40	19CS040	PRADAP K		27					54.0					N
41	19CS041	PRADEEPROSHAN R R		26					52.0					N
42	19CS042	PRAVEEN P		31					62.0					Y
43	19CS043	PRAVEEN S (15.S.2001)		30					60.0					Y
44	19CS044	PRAVEEN S (15.12.000)		32					64.0					Y
45	19CS045	PRIYA DHARSHINI S		50					100.0					Y
46	19CS046	SARAN M		48					96.0					Y
47	19CS047	SIVANESAN G		47					94.0					Y
48	19CS048	SNEGA T		49					98.0					Y
49	19CS049	SRIDHAR C		31					62.0					Y
50	19CS050	SRIDHAR S		17					34.0					N
51	19CS051	SUDEEP P		33					66.0					Y
52	19CS052	SUDHARSAN R		29					58.0					N
53	19CS053	SUREKA M		27					54.0					N
54	19CS054	SWATHI C		29					58.0					N
55	19CS055	VENKATESAN M		34					68.0					Y
56	19CS056	VIDHYASAGAR J		33					66.0					Y
57	19CS057	VIGNESH P		47					94.0					Y

58	19CS058	VIHASINI M	26					52.0					N
59	19CS059	VIMALKUMAR M	32					64.0					Y
60	19CS060	YUKENTHIRAN V	41					82.0					Y
61	19CS061	ATLA GAYATHRI	50					100.0					Y
62	19CS063	ATLA SRUTI	34					68.0					Y
63	19CS064	BOMMEPALLE HARSHA VARDHAN REDDY	38					76.0					Y
64	19CS065	BUCHI DHANUSH	48					96.0					Y
65	19CS066	C DINESH	34					68.0					Y
66	19CS067	DANDE PAVAN KALYAN	46					92.0					Y
67	19CS068	EDDAGOTTI ANIL RAJ	35					70.0					Y
68	19CS069	GNANASHANKARAN N	34					68.0					Y
69	19CS070	GOKULKUMAR M	19					38.0					N
70	19CS071	GUGGULLA HARISH REDDY	28					56.0					N
71	19CS072	K VAISHNAVI	35					70.0					Y
72	19CS073	KADURU ROHITH	33					66.0					Y
73	19CS074	KALAVAPALLI SATHISH REDDY	32					64.0					Y
74	19CS075	KARODI SURESH REDDY	47					94.0					Y
75	19CS076	KAVIYARASAN P	32					64.0					Y
76	19CS077	KODURU VISHNU VARDHAN REDDY	37					74.0					Y
77	19CS078	KOMMERLA MAHIMA	29					58.0					N
78	19CS079	KONDURU SOHITH VARMA	33					66.0					Y

79	19CS080	KOTIPALLI MANOJ KUMAR		27					54.0				N
80	19CS081	KOVURU VENKATASAI SUMANTH		34					68.0				Y
81	19CS082	KUMMARA ARAVIND		33					66.0				Y
82	19CS083	LAKSHMISETTY HEMANTH SAI		40					80.0				Y
83	19CS084	LINGI REDDY MAHANTH KUMAR REDDY		48					96.0				Y
84	19CS085	M LOKESH		32					64.0				Y
85	19CS086	MADALA VINAY		33					66.0				Y
86	19CS087	MADISETTY VEERA BHASKAR		29					58.0				N
87	19CS088	MANGALA MUNIVARDHAN		35					70.0				Y
88	19CS089	MANJU V		48					96.0				Y
89	19CS090	MIDDISUNDAR BHARATH KUMAR		32					64.0				Y
90	19CS091	MUKESH A		31					62.0				Y
91	19CS092	MUMMADI PRAVEEN KUMAR REDDY		28					56.0				N
92	19CS093	NAIDUGARI NARASIMHA NAIDU		49					98.0				Y
93	19CS094	OBULIPURUSOTHAMAN K		36					72.0				Y
94	19CS095	PATNAM BABAFAKRUDDIN		29					58.0				N
95	19CS096	PEYYALA. PRUDHVI		19					38.0				N
96	19CS097	PONAGANI KUSUMASREE		21					42.0				N
97	19CS098	PONNAM MANIKUMAR		24					48.0				N
98	19CS099	PONNAPATI DEEPAK KUMAR REDDY		34					68.0				Y
99	19CS100	PONNAPATI DINESH KUMAR REDDY		33					66.0				Y

100	19CS101	POREDDY RANJITH KUMAR REDDY		27					54.0				N
101	19CS102	POTHUPETA BHARGHAV		26					52.0				N
102	19CS103	TELLA PRASANTH KUMAR		31					62.0				Y
103	19CS104	RACHURI LOKESH		50					100.0				Y
104	19CS105	RITHIKA V		40					80.0				Y
105	19CS106	RUDRAPATI VISHNUVARDHAN		18					36.0				N
106	19CS107	SAMUKTHA M		14					28.0				N
107	19CS108	SANDHI BOINA SRINIVASULU		31					62.0				Y
108	19CS109	SARAVANAN G		34					68.0				Y
109	19CS110	SELVAKUMAR R		34					68.0				Y
110	19CS111	SHANKAVI M		28					56.0				N
111	19CS112	SIVAKUMAR S K		42					84.0				Y
112	19CS113	SOWNDHAR S		32					64.0				Y
113	19CS114	TELAGARA VISHNUKANTH REDDY		32					64.0				Y
114	19CS115	THALISETTY KISHORE		38					76.0				Y
115	19CS116	VALLURU PREETHIKA		47					94.0				Y
116	19CS117	VANDHAVASI YUVARAJU		24					48.0				N
117	19CS118	VASANTHAPRIYA N J		28					56.0				N
118	19CS119	VELPULA JAYA KRISHNA YADAV		31					62.0				Y
119	20CS501	AKASH A		39					78.0				Y
120	20CS502	DHEVA M		35					70.0				Y

121	20CS503	DHIVYA R		39					78.0					N
122	20CS504	DINESH B		49					98.0					Y
123	20CS505	HEMANTH E C		37					74.0					Y
124	20CS506	JAMUNA S		49					98.0					Y
125	20CS507	KEERTHANA T		47					94.0					Y
126	20CS508	MANIKANDAN M		40					80.0					Y
127	20CS509	MUTHUVEL A		50					100.0					Y
128	20CS510	PRAVEENKUMAR V		14					28.0					N
129	20CS511	RANJITH M		50					100.0					Y
130	20CS512	SOUNDHARYA S		34					68.0					Y
131	20CS513	SOUNDHAR S		41					82.0					Y
132	20CS514	VIMAL S		28					56.0					N
133	20CS515	K.KIRITHISWARAN		33					66.0					Y
134	20CS516	S.TAMILSELVAN		32					64.0					Y

1. 8/8/2021
2. Course Faculty 8/8/2021

HOD 8/8/21

Muthayammal Engineering College														Year/ Sem : II/IV					R2016		Viva voce(MKC)	
Department of Computer Science and Engineering														Batch :2019-2023					Total Students			134
Name of the Faculty: Mr.S.R.Sridhar & Mr.V.Karuppuchamy														No. Students Attained					107	108	96	102	89
Course Code & Name: 19CSC12 & Software Engineering														% of CO Attainment					79.9	80.6	71.6	76.1	66.4
			CO	1	2	3	4	5	6	7	8	9	10	Total	CO Attainment Level								2
S.No	Roll No.	Name of Student	CO1	1	1	1	1	1	1	1	1	1	1		CO1	CO2	CO3	CO4	CO5	CO1	CO2	CO3	CO4	CO5
			Marks	10	10	10	10	10	10	10	10	10	10	10	2	2	2	2	2	60	60	60	60	60
1	18CS117	VENMATHI G		1	1	1	1	0	0	0	0	0	0	6	100.0	100.0	50.0	50.0	0.0	Y	Y	N	N	N
2	19CS001	AKASH KUMAR M R		1	1	1	1	1	1	1	1	1	0	9	100.0	100.0	100.0	100.0	50.0	Y	Y	Y	Y	N
3	19CS002	AMANULLA M		1	1	1	1	1	1	1	0	0	0	7	100.0	100.0	50.0	100.0	0.0	Y	Y	N	Y	N
4	19CS004	ARUNKUMAR V		0	1	0	1	0	0	0	1	0	0	4	50.0	50.0	0.0	50.0	50.0	N	N	N	N	N
5	19CS005	ASWIN RAJA K		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
6	19CS006	BAVADHARINI E		1	1	1	1	1	1	0	0	0	0	6	100.0	100.0	100.0	100.0	0.0	Y	Y	Y	Y	N
7	19CS007	BAVITHRAN K		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
8	19CS008	BHARATHI V		1	1	1	1	1	1	1	1	1	0	9	100.0	100.0	100.0	100.0	50.0	Y	Y	Y	Y	N
9	19CS009	CHANDHRU G		1	1	1	1	1	1	1	1	1	0	9	100.0	100.0	100.0	100.0	50.0	Y	Y	Y	Y	N
10	19CS010	CYRILROY S		1	1	1	1	1	1	1	1	0	0	7	100.0	100.0	50.0	100.0	0.0	Y	Y	N	Y	N
11	19CS011	DANIEL SANTHOSH Y		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
12	19CS012	DEVADHARSHINI R		0	1	0	1	1	1	1	1	1	0	7	0.0	100.0	100.0	100.0	50.0	N	Y	Y	Y	N
13	19CS013	DHANEESH M		1	1	1	1	1	1	1	0	1	0	8	100.0	100.0	100.0	50.0	50.0	Y	Y	Y	N	N
14	19CS014	DHARANESH G		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
15	19CS015	ELAVARASAN S		1	1	1	0	1	1	0	0	0	1	7	100.0	50.0	100.0	50.0	50.0	Y	N	Y	N	N
16	19CS016	FOLK N		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
17	19CS017	GOBIKRISHNAN R		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
18	19CS018	GOKULA KANNAN M		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
19	19CS019	GOPI S		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
20	19CS020	GOWTHAM R		1	0	1	1	1	1	1	1	1	1	9	50.0	100.0	100.0	100.0	100.0	N	Y	Y	Y	Y
21	19CS021	GURUSARAN E		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
22	19CS022	HARIPRASATH M		1	1	1	0	1	1	0	0	1	1	7	100.0	50.0	100.0	50.0	100.0	Y	N	Y	N	Y
23	19CS023	HENDRY CHARLES G		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
24	19CS024	JAGADEESWARAN P		1	1	1	0	1	0	0	1	1	1	7	100.0	50.0	50.0	50.0	100.0	Y	N	N	N	Y

25	19CS025	JAGATH K R		1	1	0	0	1	0	1	1	0	1	6	60.0	50.0	50.0	50.0	50.0	N	N	N	N	N
26	19CS026	JAWAHARBABU R		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
27	19CS027	JAYASHANGAREE S		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
28	19CS028	JAYASURYA C		1	0	1	1	0	1	1	0	1	1	7	50.0	100.0	50.0	50.0	100.0	N	Y	N	N	Y
29	19CS029	JOSHINI M		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
30	19CS030	KARTHIKEYAN K		1	0	1	1	1	1	1	1	1	1	9	50.0	100.0	100.0	100.0	100.0	N	Y	Y	Y	Y
31	19CS031	KAVIN K R		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
32	19CS032	KAVIN R		1	1	1	1	0	1	1	1	0	1	8	50.0	100.0	50.0	50.0	100.0	N	Y	N	N	Y
33	19CS033	KUMAR M		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
34	19CS034	LOKESHWARAN V		1	0	1	1	1	1	0	1	0	1	7	50.0	50.0	100.0	50.0	50.0	N	N	Y	N	N
35	19CS035	MANIKANDAN J		1	0	1	1	1	1	1	1	0	1	9	50.0	100.0	50.0	100.0	100.0	N	Y	N	Y	Y
36	19CS036	MOHANARAMAN V S		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
37	19CS037	MUKESH KANNA R		1	1	1	1	1	1	1	1	0	1	9	100.0	100.0	100.0	100.0	50.0	Y	Y	Y	Y	N
38	19CS038	MURALITHARAN G		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
39	19CS039	NAVEEN S		1	0	1	0	1	1	1	1	1	1	8	50.0	50.0	100.0	100.0	100.0	N	N	Y	Y	Y
40	19CS040	PRADAP K		1	1	1	0	1	1	0	0	1	1	7	100.0	50.0	100.0	100.0	100.0	Y	N	Y	N	Y
41	19CS041	PRADEEPROSHAN R R		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
42	19CS042	PRAVEEN P		0	1	1	0	1	1	0	1	1	1	7	50.0	50.0	100.0	50.0	100.0	N	N	Y	N	Y
43	19CS043	PRAVEEN S (15.5.2001)		1	1	0	1	0	1	1	0	0	1	7	100.0	50.0	50.0	50.0	100.0	Y	N	N	N	Y
44	19CS044	PRAVEEN S (15.12.000)		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
45	19CS045	PRIYA DHARSHINI S		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
46	19CS046	SARAN M		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
47	19CS047	SIVANESAN G		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
48	19CS048	SNEGA T		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
49	19CS049	SRIDHAR C		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
50	19CS050	SRIDHAR S		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
51	19CS051	SUDEEP P		1	1	1	1	1	1	1	1	0	1	9	100.0	100.0	100.0	100.0	50.0	Y	Y	Y	Y	N
52	19CS052	SUDHARSAN R		1	1	1	1	1	1	1	1	0	1	9	100.0	100.0	100.0	100.0	50.0	Y	Y	Y	Y	N
53	19CS053	SUREKA M		1	1	1	1	1	1	1	1	0	1	9	100.0	100.0	100.0	100.0	50.0	Y	Y	Y	Y	N
54	19CS054	SWATHI C		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y

55	19CS055	VENKATESAN M		1	0	0	1	1	1	1	1	1	1	9	100.0	50.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
56	19CS056	VIDHYASAGAR J		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
57	19CS057	VIGNESH P		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
58	19CS058	VIHASINI M		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
59	19CS059	VIMALKUMAR M		1	0	0	1	1	1	1	0	0	0	6	100.0	50.0	100.0	100.0	100.0	Y	N	Y	Y	Y
60	19CS060	YUKENTHIRAN V		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
61	19CS061	ATLA GAYATHRI		0	0	0	1	1	1	0	0	1	0	5	0.0	50.0	100.0	100.0	50.0	N	N	Y	Y	N
62	19CS063	ATLA SRUTI		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
63	19CS064	BOMMEPALLE HARSHA VARDHAN REDDY		1	1	1	1	0	0	1	1	1	1	8	100.0	100.0	100.0	100.0	100.0	Y	Y	N	Y	Y
64	19CS065	BUCHI DHANUSH		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
65	19CS066	C DINESH		1	0	0	1	1	1	1	1	1	1	8	50.0	50.0	100.0	100.0	100.0	N	N	Y	Y	Y
66	19CS067	DANDE PAVAN KALYAN		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
67	19CS068	EDDAGOTTI ANIL RAJ		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
68	19CS069	GNANASHANKARAN N		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
69	19CS070	GOKULKUMAR M		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
70	19CS071	GUGGULLA HARISH REDDY		1	0	1	1	1	1	1	1	1	1	9	50.0	100.0	100.0	100.0	100.0	N	Y	Y	Y	Y
71	19CS072	K VAISHNAVI		1	0	1	1	0	1	1	1	1	1	8	50.0	100.0	50.0	100.0	100.0	N	Y	N	Y	Y
72	19CS073	KADURU ROHITH		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
73	19CS074	KALAVAPALLI SATHISH REDDY		1	0	0	1	1	1	1	1	1	0	8	50.0	50.0	100.0	100.0	100.0	N	N	Y	Y	Y
74	19CS075	KARODI SURESH REDDY		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
75	19CS076	KAVIYARASAN P		1	0	0	1	1	1	1	1	1	0	7	50.0	50.0	100.0	100.0	50.0	N	N	Y	Y	N
76	19CS077	KODURU VISHNU VARDHAN REDDY		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
77	19CS078	KOMMERLA MAHIMA		1	1	1	1	0	0	1	1	1	1	8	100.0	100.0	0.0	100.0	100.0	Y	Y	N	Y	Y
78	19CS079	KONDURU SOHITH VARMA		1	1	1	1	0	0	1	1	1	1	8	100.0	100.0	0.0	100.0	100.0	Y	Y	N	Y	Y
79	19CS080	KOTIPALLI MANOJ KUMAR		1	1	1	1	1	1	1	1	0	0	8	100.0	100.0	100.0	100.0	0.0	Y	Y	Y	Y	N
80	19CS081	KOVURU VENKATASAI SUMANTH		1	1	1	1	1	1	1	1	0	0	8	100.0	100.0	100.0	100.0	0.0	Y	Y	Y	Y	N
81	19CS082	KUMMARA ARAVIND		1	1	1	1	1	1	1	1	0	1	9	100.0	100.0	100.0	100.0	50.0	Y	Y	Y	Y	N
82	19CS083	LAKSHMISETTY HEMANTH SAI		1	1	1	1	1	1	1	1	0	1	9	100.0	100.0	100.0	100.0	50.0	Y	Y	Y	Y	N
83	19CS084	LINGI REDDY MAHANTH KUMAR REDDY		1	1	1	1	1	0	1	1	0	1	8	100.0	100.0	50.0	100.0	50.0	Y	Y	N	Y	N
84	19CS085	M LOKESH		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y

85	19CS086	MADALA VINAY		1	0	1	1	1	0	1	1	1	1	8	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
86	19CS087	MADISETTY VEERA BHASKAR		1	1	0	1	1	0	1	1	1	1	8	100.0	100.0	50.0	100.0	100.0	Y	Y	N	Y	Y
87	19CS088	MANGALA MUNIVARDHAN		1	1	1	1	1	0	1	0	1	1	8	100.0	100.0	50.0	50.0	100.0	Y	Y	N	Y	Y
88	19CS089	MANJU V		1	1	1	1	0	0	1	1	1	1	8	100.0	100.0	100.0	100.0	100.0	Y	Y	N	Y	Y
89	19CS090	MIDDISUNDAR BHARATH KUMAR		1	1	1	1	1	0	1	1	1	1	9	100.0	100.0	50.0	100.0	100.0	Y	Y	N	Y	Y
90	19CS091	MUKESH A		1	0	1	1	1	0	1	1	1	1	9	100.0	100.0	50.0	100.0	100.0	Y	Y	N	Y	Y
91	19CS092	MUMMADI PRAVEEN KUMAR REDDY		1	1	1	1	1	0	1	1	1	1	9	100.0	100.0	50.0	100.0	100.0	Y	Y	N	Y	Y
92	19CS093	NAIDUGARI NARASIMHA NAIDU		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
93	19CS094	OBULIPURUSOTHAMAN K		1	0	1	1	0	1	0	0	1	0	6	50.0	100.0	0.0	50.0	50.0	N	Y	N	N	N
94	19CS095	PATNAM BABAFAKRUDDIN		1	0	1	1	1	0	1	1	1	1	8	100.0	100.0	100.0	100.0	100.0	Y	Y	N	Y	Y
95	19CS096	PEYYALA. PRUDHVI		1	1	0	1	0	1	1	1	1	1	7	100.0	50.0	50.0	50.0	100.0	Y	Y	N	N	Y
96	19CS097	PONAGANI KUSUMASREE		1	1	0	1	1	1	1	1	1	1	9	100.0	100.0	100.0	100.0	100.0	Y	Y	N	Y	Y
97	19CS098	PONNAM MANIKUMAR		1	1	1	1	1	0	1	1	1	1	9	100.0	100.0	50.0	100.0	100.0	Y	Y	N	Y	Y
98	19CS099	PONNAPATI DEEPAK KUMAR REDDY		1	1	1	1	1	0	1	1	1	1	9	100.0	100.0	50.0	100.0	100.0	Y	Y	N	Y	Y
99	19CS100	PONNAPATI DINESH KUMAR REDDY		1	1	1	1	1	0	1	1	1	1	9	100.0	100.0	50.0	100.0	100.0	Y	Y	N	Y	Y
100	19CS101	POREDDY RANJITH KUMAR REDDY		1	1	1	1	1	0	1	0	0	0	6	100.0	100.0	50.0	50.0	0.0	Y	Y	N	N	N
101	19CS102	POTHUPETA BHARGHAV		1	1	1	1	1	1	1	1	1	0	9	100.0	100.0	100.0	100.0	50.0	Y	Y	Y	Y	N
102	19CS103	TELLA PRASANTH KUMAR		1	1	1	1	1	0	0	0	0	0	5	100.0	100.0	50.0	0.0	0.0	Y	Y	N	N	N
103	19CS104	RACHURI LOKESH		0	1	0	1	0	0	0	1	0	1	4	50.0	50.0	0.0	50.0	50.0	N	N	N	N	N
104	19CS105	RITHIKA V		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
105	19CS106	RUDRAPATI VISHNUVARDHAN		1	1	1	1	1	1	0	0	0	0	6	100.0	100.0	100.0	0.0	0.0	Y	Y	Y	N	N
106	19CS107	SAMUKTHA M		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
107	19CS108	SANDHI BOINA SRINIVASULU		1	1	1	1	1	1	1	1	1	0	9	100.0	100.0	100.0	100.0	50.0	Y	Y	Y	Y	N
108	19CS109	SARAVANAN G		1	1	1	1	1	1	1	1	1	0	9	100.0	100.0	100.0	100.0	50.0	Y	Y	Y	Y	N
109	19CS110	SELVAKUMAR R		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
110	19CS111	SHANKAVI M		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
111	19CS112	SIVAKUMAR S K		0	0	1	1	1	1	1	1	1	0	7	0.0	100.0	100.0	100.0	50.0	N	Y	Y	Y	N
112	19CS113	SOWNDHAR S		1	1	1	1	1	1	1	0	1	0	8	100.0	100.0	100.0	50.0	50.0	Y	Y	Y	N	N
113	19CS114	TELAGARA VISHNUKANTH REDDY		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
114	19CS115	THALISETTY KISHORE		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y

115	19CS116	VALLURU PREETHIKA		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y			Y
116	19CS117	VANDHAVASI YUVARAJU		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y			Y
117	19CS118	VASANTHAPRIYA N J		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y			Y
118	19CS119	VELPULA JAYA KRISHNA YADAV		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y			Y
119	20CS501	AKASH A		1	0	1	0	1	1	1	1	1	0	7	100.0	0.0	100.0	100.0	100.0	Y	N	Y			Y
120	20CS502	DHEVA M		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y			Y
121	20CS503	DHIVYA R		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y			Y
122	20CS504	DINESH B		1	0	1	1	1	1	1	1	0	0	6	100.0	0.0	100.0	100.0	0.0	Y	N	Y			N
123	20CS505	HEMANTH E C		1	1	1	1	1	1	1	1	1	0	9	100.0	100.0	100.0	100.0	100.0	Y	Y	Y			N
124	20CS506	JAMUNA S		1	1	1	1	1	0	1	1	0	0	7	100.0	100.0	100.0	100.0	0.0	Y	Y	Y			N
125	20CS507	KEERTHANA T		0	1	0	1	0	0	0	1	0	1	4	0.0	100.0	0.0	100.0	0.0	N	Y	N			N
126	20CS508	MANIKANDAN M		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y			Y
127	20CS509	MUTHUVEL A		1	1	1	1	1	1	0	0	0	0	6	100.0	100.0	100.0	100.0	0.0	Y	Y	Y			N
128	20CS510	PRAVEENKUMAR V		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y			Y
129	20CS511	RANJITH M		1	1	1	1	1	1	1	1	1	0	9	100.0	100.0	100.0	100.0	50.0	Y	Y	Y			N
130	20CS512	SOUNDHARYA S		1	1	1	1	1	1	1	1	1	0	9	100.0	100.0	100.0	100.0	50.0	Y	Y	Y			N
131	20CS513	SOUNDHAR S		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y			Y
132	20CS514	VIMAL S		1	1	1	1	1	1	1	1	1	1	10	100.0	100.0	100.0	100.0	100.0	Y	Y	Y			Y
133	20CS515	K.KIRITHISWARAN		0	0	1	1	1	1	1	1	1	0	7	0.0	100.0	100.0	100.0	50.0	N	Y	Y			N
134	20CS516	S.TAMILSELVAN		1	1	1	1	1	1	1	0	1	0	8	100.0	100.0	100.0	50.0	50.0	Y	Y	Y	N		N

1.

Course Faculty

HOD

Muthayammal Engineering College

Department of Computer Science and Engineering

Name of the Faculty: Mr.S.R.Sridhar & Mr.V.Karuppuchamy

Course Code & Name: 19CSC12 & Software Engineering

II/D | R2016 | CIA-III | Batch : 2019-2023 | Total Students : 134 | No. Students Attained | 79 | 79 | 75 | 76 | 82 | % of CO Attainment | 59.0 | 59.0 | 56.0 | 56.7 | 61.2 | CO Attainment Level | 1 | 1 | 1 | 1 | 2

S.No	Roll No.	Name of Student	Q No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Total		CO1	CO2	CO3	CO4	CO5
			CO No.	1	1	2	2	3	3	4	4	5	5	1	2	3	4	5							
			Marks	2	2	2	2	2	2	2	2	2	2	16	16	16	16	16	100		8	32	40	20	20
1	18CS117	VENMATHI G		2	2	0	2	2	2	2	2	2	2	14	10	8	12	7	71		8.0	62.5	60.0	50.0	55.0						
2	19CS001	AKASH KUMAR M R		2	2	2	2	2	2	2	2	2	2	8	8	8	12	4	65		8.0	62.5	60.0	45.0	50.0						
3	19CS002	AMANULLA M		2	2	2	2	2	2	2	2	2	2	8	8	8	8	4	57		8.0	62.5	50.0	45.0	50.0						
4	19CS004	ARUNKUMAR V		2	2	2	2	2	2	2	2	2	2	8	8	8	8	4	49		8.0	62.5	50.0	45.0	50.0						
5	19CS005	ASWIN RAJA K		2	2	2	2	2	2	2	2	2	2	8	8	8	8	4	40		8.0	62.5	50.0	45.0	50.0						
6	19CS006	BAVADHARINI E		2	2	2	2	2	2	2	2	2	2	14	14	14	14	16	96		8.0	87.5	90.0	70.0	90.0						
7	19CS007	BAVITHRAN K		2	2	2	2	2	2	2	2	2	2	16	16	16	16	16	96		8.0	100.0	100.0	80.0	90.0						
8	19CS008	BHARATHI V		2	2	2	2	2	2	2	2	2	2	14	14	16	16	16	91		8.0	87.5	90.0	75.0	90.0						
9	19CS009	CHANDHRU G		2	1	1	1	2	1	0	0	2	2	8	8	8	8	8	46		8.0	50.0	50.0	40.0	40.0						
10	19CS010	CYRILROY S		2	1	1	2	2	2	2	2	2	2	0	12	16	8	8	50		8.0	62.5	50.0	35.0	40.0						
11	19CS011	DANIEL SANTHOSH Y		2	2	1	2	2	2	2	2	2	2	16	16	16	8	16	73		8.0	75.0	80.0	50.0	80.0						
12	19CS012	DEVADHARSHINI R		0	2	0	1	1	0	0	1	1	2	1	14	14	12	0	62		8.0	75.0	60.0	50.0	50.0						
13	19CS013	DHANEESH M		2	2	2	2	2	2	2	2	2	2	16	16	16	16	16	75		8.0	87.5	90.0	70.0	80.0						
14	19CS014	DHARANESH G		2	2	2	2	2	0	2	2	2	0	16	14	8	8	8	47		8.0	87.5	80.0	55.0	80.0						
15	19CS015	ELAVARASAN S		2	2	0	1	2	0	1	2	2	2	16	8	16	8	8	43		8.0	75.0	70.0	50.0	70.0						
16	19CS016	FOLK N		1	0	1	1	2	2	2	2	2	2	12	12	0	0	4	26		8.0	62.5	40.0	30.0	40.0						
17	19CS017	GOBIKRISHNAN R		2	2	0	2	2	2	0	1	0	2	0	14	14	14	14	66		8.0	87.5	80.0	55.0	70.0						
18	19CS018	GOKULA KANNAN M		2	2	2	2	2	2	2	2	2	2	14	8	16	8	8	68		8.0	87.5	80.0	60.0	70.0						
19	19CS019	GOPI S		2	2	1	2	0	2	2	1	2	2	12	12	8	8	8	49		8.0	75.0	70.0	50.0	60.0						
20	19CS020	GOWTHAM R		2	2	2	0	0	0	2	1	2	0	6	7	5	6	0	37		8.0	56.3	50.0	35.0	40.0						
21	19CS021	GURUSARAN E		2	1	1	2	1	1	2	1	2	2	8	6	7	14	10	67		8.0	68.8	70.0	50.0	70.0						
22	19CS022	HARIPRASATH M		2	2	1	2	0	2	1	1	2	2	8	7	0	5	6	41		8.0	56.3	50.0	35.0	40.0						
23	19CS023	HENDRY CHARLES G		2	2	0	2	1	1	0	1	0	2	0	14	14	14	13	66		8.0	81.3	80.0	55.0	75.0						
24	19CS024	JAGADEESWARAN P		2	2	0	0	2	2	2	2	2	2	15	8	6	5	15	69		8.0	93.8	90.0	65.0	95.0						
25	19CS025	JAGATH K R		2	2	1	1	2	1	2	0	1	2	9	8	7	6	5	49		8.0	68.8	50.0	40.0	40.0						
26	19CS026	JAWAHARBABU R		2	2	2	1	2	2	2	1	2	2	14	16	15	16	14	93		8.0	93.8	90.0	65.0	90.0						

27	19CS027	JAYASHANGAREE S																91											
28	19CS028	JAYASURYA C																49											
29	19CS029	JOSHINI M																77											
30	19CS030	KARTHIKEYAN K																65											
31	19CS031	KAVIN K R																53											
32	19CS032	KAVIN R																45											
33	19CS033	KUMAR M																82											
34	19CS034	LOKESHWARAN V																57											
35	19CS035	MANIKANDAN J																84											
36	19CS036	MOHANARAMAN V S																75											
37	19CS037	MUKESH KANNA R																62											
38	19CS038	MURALITHARAN G																62											
39	19CS039	NAVEEN S																97											
40	19CS040	PRADAP K																51											
41	19CS041	PRADEEPROSHAN R R																50											
42	19CS042	PRAVEEN P																47											
43	19CS043	PRAVEEN S (15.5.2001)																47											
44	19CS044	PRAVEEN S (15.12.000)																85											
45	19CS045	PRIYA DHARSHINI S																93											
46	19CS046	SARAN M																68											
47	19CS047	SIVANESAN G																56											
48	19CS048	SNEGA T																88											
49	19CS049	SRIDHAR C																44											
50	19CS050	SRIDHAR S																46											
51	19CS051	SUDEEP P																65											
52	19CS052	SUDHARSAN R																66											
53	19CS053	SUREKA M																74											
54	19CS054	SWATHI C																99											
55	19CS055	VENKATESAN M																52											
56	19CS056	VIDHYASAGAR J																56											
57	19CS057	VIGNESH P																77											
58	19CS058	VIHASINI M																42											

59	19CS059	VIMALKUMAR M															43												
60	19CS060	YUKENTHIRAN V															51												
61	19CS061	ATLA GAYATHRI															87												
62	19CS063	ATLA SRUTI															85												
63	19CS064	BOMMEPALLE HARSHA VARDHAN REDDY															50												
64	19CS065	BUCHI DHANUSH															95												
65	19CS066	C DINESH															62												
66	19CS067	DANDE PAVAN KALYAN															81												
67	19CS068	EDDAGOTTI ANIL RAJ															71												
68	19CS069	GNANASHANKARAN N															51												
69	19CS070	GOKULKUMAR M															62												
70	19CS071	GUGGULLA HARISH REDDY															83												
71	19CS072	K VAISHNAVI															84												
72	19CS073	KADURU ROHITH															39												
73	19CS074	KALAVAPALLI SATHISH REDDY															45												
74	19CS075	KARODI SURESH REDDY															47												
75	19CS076	KAVIYARASAN P															93												
76	19CS077	KODURU VISHNU VARDHAN REDDY															91												
77	19CS078	KOMMERLA MAHIMA															95												
78	19CS079	KONDURU SOHITH VARMA															45												
79	19CS080	KOTIPALLI MANOJ KUMAR															43												
80	19CS081	KOVURU VENKATASAI SUMANTH															73												
81	19CS082	KUMMARA ARAVIND															39												
82	19CS083	LAKSHMISETTY HEMANTH SAI															90												
83	19CS084	LINGI REDDY MAHANTH KUMAR REDDY															92												
84	19CS085	M LOKESH															80												
85	19CS086	MADALA VINAY															90												
86	19CS087	MADISETTY VEERA BHASKAR															46												
87	19CS088	MANGALA MUNIVARDHAN															38												
88	19CS089	MANJU V															90												
89	19CS090	MIDDISUNDAR BHARATH KUMAR															43												
90	19CS091	MUKESH A															92												

91	19CS092	MUMMADI PRAVEEN KUMAR REDDY		1	1	0	2	0	1	0	2	2	2	14	10	12	13	16	74	74.0	82.2	80.0	85.0	90.0	Y	Y	Y	Y	Y
92	19CS093	NAIDUGARI NARASIMHA NAIDU		1	1	2	2	1	2	1	2	0	2	15	14	16	16	16	89	89.0	98.8	95.0	90.0	100.0	Y	Y	Y	Y	Y
93	19CS094	OBULIPURUSOTHAMAN K		2	1	2	2	1	2	0	0	2	2	14	10	14	15	16	83	83.0	92.2	90.0	75.0	95.0	Y	Y	Y	Y	Y
94	19CS095	PATNAM BABAFAKRUDDIN		2	2	2	2	2	0	1	0	0	2	5	5	5	5	2	31	31.0	34.4	40.0	10.0	40.0	N	N	N	N	N
95	19CS096	PEYYALA. PRUDHVI		2	2	1	2	2	2	1	2	2	2	15	14	13	12	16	88	88.0	97.7	95.0	80.0	95.0	Y	Y	Y	Y	Y
96	19CS097	PONAGANI KUSUMASREE		2	1	1	2	2	2	2	0	2	2	12	11	12	12	12	53	53.0	58.8	65.0	50.0	65.0	N	N	N	N	N
97	19CS098	PONNAM MANIKUMAR		1	1	2	2	2	2	2	2	2	2	13	12	10	12	14	59	59.0	65.5	65.0	60.0	70.0	N	N	N	N	N
98	19CS099	PONNAPATI DEEPAK KUMAR REDDY		2	2	2	2	2	2	2	2	1	2	11	10	14	12	16	76	76.0	84.4	80.0	70.0	100.0	Y	Y	Y	Y	Y
99	19CS100	PONNAPATI DINESH KUMAR REDDY		1	2	2	2	2	2	2	2	2	2	12	15	14	13	16	87	87.0	96.6	95.0	80.0	95.0	Y	Y	Y	Y	Y
100	19CS101	POREDDY RANJITH KUMAR REDDY		2	2	1	2	1	2	2	2	0	2	4	5	5	5	5	33	33.0	36.6	35.0	35.0	40.0	N	N	N	N	N
101	19CS102	POTHUPETA BHARGHAV		2	2	2	2	2	2	1	2	2	2	15	14	13	13	15	83	83.0	92.2	90.0	75.0	95.0	Y	Y	Y	Y	Y
102	19CS103	TELLA PRASANTH KUMAR		1	0	2	2	2	2	2	2	2	2	16	16	12	12	16	88	88.0	97.7	100.0	80.0	95.0	Y	Y	Y	Y	Y
103	19CS104	RACHURI LOKESH		2	2	2	2	1	2	1	2	1	2	8	9	7	8	9	58	58.0	64.4	60.0	55.0	60.0	N	N	N	N	Y
104	19CS105	RITHIKA V		2	2	2	2	2	2	2	2	2	2	16	16	16	16	16	99	99.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
105	19CS106	RUDRAPATI VISHNUVARDHAN		2	2	2	2	0	1	0	2	0	1	8	7	5	6	9	47	47.0	52.2	50.0	40.0	60.0	N	N	N	N	N
106	19CS107	SAMUKTHA M		2	2	2	2	2	2	2	2	2	2	10	15	14	15	14	68	68.0	75.5	70.0	65.0	80.0	N	N	N	Y	Y
107	19CS108	SANDHI BOINA SRINIVASULU		2	1	2	0	2	1	2	2	2	2	8	9	7	5	16	61	61.0	67.7	70.0	45.0	100.0	N	N	N	N	Y
108	19CS109	SARAVANAN G		2	1	1	0	2	1	2	2	1	2	5	7	8	5	0	49	49.0	54.4	55.0	35.0	50.0	N	N	N	Y	N
109	19CS110	SELVAKUMAR R		2	2	2	2	1	2	1	2	1	2	8	9	7	5	14	60	60.0	66.6	65.0	40.0	85.0	N	Y	N	N	Y
110	19CS111	SHANKAVI M		2	2	1	2	1	2	2	2	2	2	12	14	15	16	15	93	93.0	103.3	100.0	80.0	95.0	Y	Y	Y	Y	Y
111	19CS112	SIVAKUMAR S K		0	2	0	1	1	2	2	2	2	2	14	16	16	13	16	89	89.0	98.8	95.0	85.0	100.0	Y	Y	Y	Y	Y
112	19CS113	SOWNDHAR S		2	1	2	2	2	2	2	1	2	2	15	15	12	13	16	90	90.0	100.0	90.0	80.0	100.0	Y	Y	Y	Y	Y
113	19CS114	TELAGARA VISHNUKANTH REDDY		2	2	2	2	2	2	2	0	0	1	8	9	7	5	16	60	60.0	66.6	65.0	45.0	85.0	N	Y	N	N	Y
114	19CS115	THALISETTY KISHORE		2	2	0	2	1	2	2	1	2	2	15	16	16	15	15	89	89.0	98.8	95.0	85.0	95.0	Y	Y	Y	Y	Y
115	19CS116	VALLURU PREETHIKA		1	0	1	1	2	2	2	2	2	2	12	13	15	14	15	84	84.0	93.3	95.0	80.0	95.0	Y	Y	Y	Y	Y
116	19CS117	VANDHAVASI YUVARAJU		2	2	2	2	2	2	2	2	2	2	15	16	16	14	16	97	97.0	107.7	100.0	90.0	100.0	Y	Y	Y	Y	Y
117	19CS118	VASANTHAPRIYA N J		2	2	2	2	2	2	2	2	2	2	14	13	16	16	15	94	94.0	104.4	100.0	100.0	95.0	Y	Y	Y	Y	Y
118	19CS119	VELPULA JAYA KRISHNA YADAV		2	2	2	2	2	2	2	1	2	2	15	15	15	15	15	94	94.0	104.4	95.5	95.0	90.0	95.0	Y	Y	Y	Y	Y
119	20CS501	AKASH A		2	2	2	2	2	2	2	2	1	1	14	12	13	15	12	84	84.0	93.3	81.8	85.0	70.0	Y	Y	Y	Y	Y
120	20CS502	DHEVA M		2	1	1	2	1	2	2	1	2	2	14	9	9	8	16	72	72.0	80.0	90.9	55.0	100.0	Y	N	Y	N	Y
121	20CS503	DHIVYA R		2	2	1	2	0	0	0	0	2	2	0	2	5	0	0	18	18.0	20.0	18.1	0.0	20.0	N	N	N	N	N
122	20CS504	DINESH B		2	2	1	2	1	2	2	2	1	2	16	16	11	12	14	86	86.0	95.5	90.9	80.0	85.0	Y	Y	Y	Y	Y

123	20CS505	HEMANTH E C		[redacted]	65	[redacted]
124	20CS506	JAMUNA S		[redacted]	80	[redacted]
125	20CS507	KEERTHANA T		[redacted]	48	[redacted]
126	20CS508	MANIKANDAN M		[redacted]	40	[redacted]
127	20CS509	MUTHUVEL A		[redacted]	61	[redacted]
128	20CS510	PRAVEENKUMAR V		[redacted]	96	[redacted]
129	20CS511	RANJITH M		[redacted]	91	[redacted]
130	20CS512	SOUNDHARYA S		[redacted]	69	[redacted]
131	20CS513	SOUNDHAR S		[redacted]	76	[redacted]
132	20CS514	VIMAL S		[redacted]	59	[redacted]
133	20CS515	K.KIRITHISWARAN		[redacted]	62	[redacted]
134	20CS516	S.TAMILSELVAN		[redacted]	75	[redacted]

1. 13/5/2021
2. 13/5/2021
Course Faculty

HOD
13/5

Muthayammal Engineering College																			Year/ Sem : II/IV		R2016	End Sem Exam			
Department of Computer Science and Engineering																			Batch : 2019-2023		Total Students		134		
Name of the Faculty: Mr.S.R.Sridhar & Mr.V.Karuppuchamy																			No. Students Attained		96	84	82	69	89
Course Code & Name: 19CSC12 & Software Engineering																			% of CO Attainment		71.6	62.7	61.2	51.5	66.4
S.No	Roll No.	Name of Student																	CO Attainment Level		2	2	2	1	2
			CO No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Total		CO1	CO2	CO3	CO4	CO5	CO1	CO2	CO1	CO4	CO5
			Max.	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	100		20	20	20	20	20	60	60	60	60	60
1	18CS117	VENMATHI G		3	2	3	3	3	1	2	3	3	3	3	3	3	2	2	54		60.0	60.0	60.0	60.0	60.0	Y	Y	Y	Y	Y
2	19CS001	AKASH KUMAR M R		2	2	3	2	2	3	1	2	3	3	3	3	3	3	3	60		60.0	60.0	60.0	60.0	60.0	Y	Y	Y	Y	Y
3	19CS002	AMANULLA M		2	2	3	3	3	2	3	2	2	2	2	2	2	2	2	50		50.0	60.0	60.0	60.0	60.0	N	Y	N	Y	Y
4	19CS004	ARUNKUMAR V		2	2	2	2	2	1	2	3	2	2	2	2	2	2	2	49		50.0	60.0	60.0	60.0	60.0	N	Y	N	Y	Y
5	19CS005	ASWIN RAJA K		2	2	3	2	2	3	2	2	2	2	2	2	2	2	2	45		50.0	60.0	60.0	60.0	60.0	N	Y	N	Y	Y
6	19CS006	BAVADHARINI E		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	98		100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
7	19CS007	BAVITHRAN K		2	2	2	2	2	1	1	3	2	2	2	2	2	2	2	54		60.0	60.0	60.0	60.0	60.0	Y	Y	Y	Y	Y
8	19CS008	BHARATHI V		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	62		60.0	60.0	60.0	60.0	60.0	Y	Y	Y	Y	Y
9	19CS009	CHANDHRU G		2	2	2	2	2	2	1	1	3	2	2	2	2	2	2	45		50.0	60.0	60.0	60.0	60.0	N	Y	Y	Y	Y
10	19CS010	CYRILROY S		2	2	2	2	2	2	1	2	3	2	2	2	2	2	2	47		50.0	60.0	60.0	60.0	60.0	N	Y	Y	Y	Y
11	19CS011	DANIEL SANTHOSH Y		3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	57		60.0	60.0	60.0	60.0	60.0	Y	Y	Y	Y	Y
12	19CS012	DEVADHARSHINI R		3	3	3	3	3	3	2	3	3	3	3	3	3	3	3	87		90.0	90.0	90.0	90.0	90.0	Y	Y	Y	Y	Y
13	19CS013	DHANEESH M		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	87		90.0	90.0	90.0	90.0	90.0	Y	Y	Y	Y	Y
14	19CS014	DHARANESH G		2	2	2	2	2	3	2	2	2	3	3	3	3	3	3	45		50.0	60.0	60.0	60.0	60.0	N	Y	Y	Y	Y
15	19CS015	ELAVARASAN S		2	2	3	3	3	3	2	3	3	3	3	3	3	3	3	49		50.0	60.0	60.0	60.0	60.0	N	Y	Y	Y	Y
16	19CS016	FOLK N		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	73		90.0	90.0	90.0	90.0	90.0	Y	Y	Y	Y	Y
17	19CS017	GOBIKRISHNAN R		2	2	3	3	3	2	2	3	3	3	3	3	3	3	3	67		80.0	80.0	90.0	90.0	90.0	Y	Y	Y	Y	Y
18	19CS018	GOKULA KANNAN M		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	97		100.0	100.0	100.0	100.0	100.0	Y	Y	Y	Y	Y
19	19CS019	GOPI S		2	2	3	3	3	2	3	3	3	3	3	3	3	3	3	51		60.0	60.0	60.0	60.0	60.0	Y	Y	Y	Y	Y
20	19CS020	GOWTHAM R		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	48		50.0	60.0	60.0	60.0	60.0	N	Y	Y	Y	Y
21	19CS021	GURUSARAN E		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	68		80.0	80.0	80.0	80.0	80.0	Y	Y	Y	Y	Y
22	19CS022	HARIPRASATH M		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	46		50.0	60.0	60.0	60.0	60.0	Y	Y	Y	Y	Y
23	19CS023	HENDRY CHARLES G		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	77		90.0	90.0	90.0	90.0	90.0	Y	Y	Y	Y	Y
24	19CS024	JAGADEESWARAN P		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	64		80.0	80.0	80.0	80.0	80.0	Y	Y	Y	Y	Y
25	19CS025	JAGATH K R		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	53		60.0	60.0	60.0	60.0	60.0	N	Y	N	Y	Y
26	19CS026	JAWAHARBABU R		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	66		80.0	80.0	80.0	80.0	80.0	Y	Y	Y	Y	Y
27	19CS027	JAYASHANGAREE S		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	82		80.0	80.0	80.0	80.0	80.0	Y	Y	Y	Y	Y
28	19CS028	JAYASURYA C		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	51		60.0	60.0	60.0	60.0	60.0	Y	Y	Y	Y	Y
29	19CS029	JOSHINI M		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	48		60.0	60.0	60.0	60.0	60.0	Y	Y	Y	Y	Y

30	19CS030	KARTHIKEYAN K		2	1	2	2	2	1	1	0	2	2	8	9	5	6	2	59	60.0	60.0	60.0	60.0	70.0	N	N	N	N	N
31	19CS031	KAVIN K R		2	2	2	2	2	0	2	0	2	2	6	6	6	6	6	52	65.0	65.0	65.0	60.0	80.0	Y	Y	Y	Y	Y
32	19CS032	KAVIN R		2	2	2	2	2	0	0	0	2	2	2	2	2	2	2	50	50.0	50.0	50.0	50.0	50.0	N	N	N	N	N
33	19CS033	KUMAR M		2	2	2	2	2	2	0	1	2	2	7	7	8	6	2	65	60.0	55.0	60.0	60.0	100.0	Y	Y	Y	Y	Y
34	19CS034	LOKESHWARAN V		2	2	2	1	2	2	0	0	2	0	2	2	2	2	2	54	60.0	60.0	55.0	60.0	60.0	Y	Y	Y	Y	Y
35	19CS035	MANIKANDAN J		2	1	2	2	2	2	1	2	2	2	3	6	4	4	2	48	45.0	50.0	50.0	45.0	70.0	N	N	N	N	Y
36	19CS036	MOHANARAMAN V S		2	2	2	2	2	0	2	2	2	2	4	4	2	2	2	58	60.0	60.0	60.0	60.0	80.0	Y	Y	Y	Y	Y
37	19CS037	MUKESH KANNA R		2	2	2	1	2	2	2	2	2	2	9	8	10	10	5	75	60.0	60.0	60.0	100.0	95.0	Y	Y	Y	Y	Y
38	19CS038	MURALITHARAN G		2	2	2	2	2	0	2	1	2	2	10	10	10	10	2	73	60.0	60.0	60.0	60.0	90.0	Y	Y	Y	Y	Y
39	19CS039	NAVEEN S		2	2	2	2	2	2	2	2	10	10	15	15	10	10	10	98	95.0	100.0	95.0	100.0	100.0	Y	Y	Y	Y	Y
40	19CS040	PRADAP K		2	1	2	2	2	1	2	2	2	2	10	8	8	8	2	49	50.0	50.0	55.0	55.0	60.0	N	N	N	N	Y
41	19CS041	PRADEEPROSHAN R R		2	2	2	2	2	1	2	2	2	2	5	6	8	4	10	55	65.0	60.0	65.0	65.0	70.0	Y	Y	Y	Y	Y
42	19CS042	PRAVEEN P		2	2	2	2	2	2	2	2	0	2	10	9	12	13	10	62	60.0	60.0	60.0	65.0	65.0	N	N	N	N	N
43	19CS043	PRAVEEN S (15.5.2001)		2	1	2	2	2	0	2	2	1	1	2	8	5	6	2	50	60.0	60.0	45.0	60.0	45.0	N	N	N	N	N
44	19CS044	PRAVEEN S (15.12.000)		2	2	1	1	1	0	1	2	2	2	9	8	15	9	15	70	65.0	50.0	80.0	60.0	95.0	Y	N	Y	Y	Y
45	19CS045	PRIYA DHARSHINI S		2	2	1	2	2	2	2	2	2	2	15	16	16	16	15	97	95.0	95.0	100.0	100.0	95.0	Y	Y	Y	Y	Y
46	19CS046	SARAN M		2	2	2	0	2	2	2	2	2	2	9	8	5	6	4	50	65.0	50.0	45.0	50.0	40.0	N	N	N	N	N
47	19CS047	SIVANESAN G		2	2	1	1	0	2	1	2	1	2	5	6	9	10	10	54	45.0	50.0	55.0	65.0	65.0	N	N	N	Y	Y
48	19CS048	SNEGA T		2	2	2	2	2	0	1	0	1	2	16	15	16	14	15	90	100.0	95.0	95.0	90.0	90.0	Y	Y	Y	Y	Y
49	19CS049	SRIDHAR C		2	2	2	2	2	1	2	2	2	2	4	12	16	11	10	83	95.0	90.0	95.0	95.0	90.0	Y	Y	Y	Y	Y
50	19CS050	SRIDHAR S		2	2	2	1	0	1	0	0	2	2	8	7	6	9	5	47	60.0	55.0	55.0	45.0	45.0	N	N	N	N	N
51	19CS051	SUDEEP P		2	2	1	0	0	1	0	1	2	2	3	6	7	10	10	43	45.0	55.0	50.0	65.0	60.0	N	N	N	N	Y
52	19CS052	SUDHARSAN R		2	2	1	1	2	2	1	1	0	1	9	9	12	14	15	71	60.0	65.0	80.0	80.0	80.0	Y	Y	Y	Y	Y
53	19CS053	SUREKA M		2	2	2	2	2	2	0	1	0	2	16	10	8	10	16	75	100.0	70.0	60.0	55.0	90.0	Y	Y	Y	N	Y
54	19CS054	SWATHI C		2	2	0	1	0	1	2	2	2	2	15	8	9	9	10	65	95.0	80.0	50.0	55.0	70.0	Y	Y	Y	Y	Y
55	19CS055	VENKATESAN M		1	2	2	2	2	2	0	1	0	2	8	9	8	6	5	50	50.0	50.0	60.0	55.0	85.0	Y	Y	Y	N	N
56	19CS056	VIDHYASAGAR J		1	2	2	2	1	1	1	0	2	2	15	8	9	8	16	70	90.0	60.0	55.0	45.0	100.0	Y	Y	Y	Y	Y
57	19CS057	VIGNESH P		2	2	1	0	0	0	0	2	1	2	12	8	9	7	14	60	80.0	75.0	55.0	45.0	85.0	N	N	N	N	Y
58	19CS058	VIHASINI M		2	2	2	2	0	2	1	2	2	2	8	7	5	6	2	45	60.0	55.0	65.0	45.0	80.0	N	N	N	N	N
59	19CS059	VIMALKUMAR M		2	2	2	1	2	1	1	1	0	1	5	6	8	9	0	41	45.0	45.0	55.0	55.0	5.0	N	N	N	N	N
60	19CS060	YUKENTHIRAN V		2	1	2	1	2	2	2	2	1	2	5	8	9	6	10	55	40.0	55.0	65.0	60.0	65.0	N	N	Y	N	Y
61	19CS061	ATLA GAYATHRI		2	2	2	2	2	2	1	1	2	2	9	10	12	12	16	77	65.0	70.0	80.0	70.0	100.0	Y	Y	Y	Y	Y
62	19CS063	ATLA SRUTI		2	1	2	2	2	2	2	2	2	2	10	12	13	14	15	83	65.0	80.0	85.0	90.0	95.0	Y	Y	Y	Y	Y
63	19CS064	BOMMEPALLE HARSHA VARDHAN REDDY		1	2	2	1	2	1	0	1	0	2	8	9	5	6	2	42	55.0	60.0	40.0	55.0	20.0	N	Y	N	N	N

64	19CS065	BUCHI DHANUSH		1	2	2	2	2	2	2	2	2	1	0	1	1	0	1	1	30	60.0	60.0	60.0	60.0	60.0	N	N	N	N	N
65	19CS066	C DINESH		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	38	80.0	80.0	80.0	80.0	80.0	Y	Y	Y	Y	Y
66	19CS067	DANDE PAVAN KALYAN		1	2	1	2	2	1	0	2	2	2	2	2	2	2	2	2	31	65.0	65.0	65.0	65.0	65.0	Y	Y	Y	Y	Y
67	19CS068	EDDAGOTTI ANIL RAJ		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	36	75.0	75.0	75.0	75.0	75.0	Y	Y	Y	Y	Y
68	19CS069	GNANASHANKARAN N		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	36	75.0	75.0	75.0	75.0	75.0	Y	Y	Y	Y	Y
69	19CS070	GOKULKUMAR M		2	2	2	2	0	0	2	2	0	2	2	2	2	2	2	2	32	65.0	65.0	50.0	65.0	50.0	Y	Y	N	Y	N
70	19CS071	GUGGULLA HARISH REDDY		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	36	75.0	75.0	75.0	75.0	75.0	Y	Y	Y	Y	Y
71	19CS072	K VAISHNAVI		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	36	75.0	75.0	75.0	75.0	75.0	Y	Y	Y	Y	Y
72	19CS073	KADURU ROHITH		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	36	75.0	75.0	75.0	75.0	75.0	Y	Y	Y	Y	Y
73	19CS074	KALAVAPALLI SATHISH REDDY		2	2	0	2	2	2	2	2	2	2	2	2	2	2	2	2	34	70.0	70.0	70.0	70.0	70.0	Y	Y	Y	Y	Y
74	19CS075	KARODI SURESH REDDY		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	36	75.0	75.0	75.0	75.0	75.0	Y	Y	Y	Y	Y
75	19CS076	KAVIYARASAN P		2	1	2	2	1	1	2	2	2	2	2	2	2	2	2	2	33	70.0	60.0	65.0	70.0	70.0	Y	Y	Y	Y	Y
76	19CS077	KODURU VISHNU VARDHAN REDDY		1	1	2	2	2	2	1	2	2	2	2	2	2	2	2	2	31	65.0	70.0	65.0	65.0	65.0	Y	Y	Y	Y	Y
77	19CS078	KOMMERLA MAHIMA		1	2	2	2	2	1	2	1	2	2	2	2	2	2	2	2	34	70.0	75.0	75.0	75.0	75.0	Y	Y	Y	Y	Y
78	19CS079	KONDURU SOHITH VARMA		2	2	2	2	0	1	0	2	1	2	2	2	2	2	2	2	30	60.0	65.0	65.0	65.0	60.0	N	Y	Y	N	N
79	19CS080	KOTIPALLI MANOJ KUMAR		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	36	75.0	75.0	75.0	60.0	60.0	N	N	N	N	Y
80	19CS081	KOVURU VENKATASAI SUMANTH		2	2	2	0	2	2	2	2	2	2	16	15	15	14	15	15	93	100.0	95.0	95.0	90.0	95.0	Y	Y	Y	Y	Y
81	19CS082	KUMMARA ARAVIND		2	2	2	2	1	1	2	2	1	2	8	15	14	15	15	15	84	90.0	95.0	90.0	85.0	90.0	Y	Y	Y	Y	Y
82	19CS083	LAKSHMISETTY HEMANTH SAI		2	2	2	2	2	2	1	2	2	2	15	14	15	16	16	15	95	95.0	90.0	95.0	95.0	100.0	Y	Y	Y	Y	Y
83	19CS084	LINGI REDDY MAHANTH KUMAR REDDY		1	2	2	1	2	2	0	2	2	2	14	12	13	7	6	6	68	85.0	75.0	85.0	85.0	70.0	Y	Y	Y	N	N
84	19CS085	M LOKESH		2	1	2	1	0	2	1	0	1	1	15	14	9	14	14	14	71	90.0	85.0	85.0	85.0	85.0	Y	N	N	Y	Y
85	19CS086	MADALA VINAY		2	2	2	2	2	2	2	2	2	2	15	14	9	16	15	15	83	85.0	80.0	85.0	90.0	85.0	Y	Y	Y	Y	Y
86	19CS087	MADISETTY VEERA BHASKAR		2	2	1	1	0	2	1	0	1	2	16	14	12	5	8	8	44	80.0	80.0	85.0	80.0	85.0	N	N	N	N	N
87	19CS088	MANGALA MUNIVARDHAN		2	1	2	1	2	2	2	1	1	2	12	14	9	5	0	1	43	80.0	85.0	85.0	80.0	85.0	N	N	N	N	N
88	19CS089	MANJU V		2	1	2	2	2	2	2	2	1	2	15	14	15	14	10	12	86	90.0	90.0	85.0	90.0	85.0	Y	Y	Y	Y	Y
89	19CS090	MIDDISUNDAR BHARATH KUMAR		2	2	2	2	0	0	1	2	1	2	10	14	9	15	9	10	45	70.0	70.0	75.0	80.0	60.0	N	Y	N	N	Y
90	19CS091	MUKESH A		2	2	2	2	2	2	2	2	2	1	14	15	16	15	10	12	89	100.0	90.0	90.0	85.0	85.0	Y	Y	Y	Y	Y
91	19CS092	MUMMADI PRAVEEN KUMAR REDDY		2	2	2	2	2	2	2	2	2	2	15	10	15	5	15	15	80	95.0	70.0	95.0	45.0	95.0	Y	Y	Y	N	Y
92	19CS093	NAIDUGARI NARASIMHA NAIDU		2	2	2	1	2	2	2	2	1	2	16	15	14	16	16	15	95	100.0	90.0	85.0	100.0	95.0	Y	Y	Y	Y	Y
93	19CS094	OBULIPURUSOTHAMAN K		2	2	2	1	2	1	2	2	1	2	16	15	14	8	9	10	79	100.0	90.0	85.0	60.0	60.0	Y	Y	Y	Y	Y
94	19CS095	PATNAM BABAFAKRUDDIN		2	2	2	1	0	1	0	0	2	0	8	9	9	8	9	7	53	60.0	60.0	50.0	40.0	55.0	Y	Y	N	N	N
95	19CS096	PEYYALA. PRUDHVI		2	2	2	2	0	2	2	1	1	2	15	14	9	7	15	15	70	95.0	60.0	55.0	90.0	90.0	Y	Y	N	N	Y
96	19CS097	PONAGANI KUSUMASREE		2	2	2	2	2	1	2	2	2	2	10	16	16	15	16	16	92	70.0	100.0	95.0	55.0	100.0	Y	Y	Y	Y	Y
97	19CS098	PONNAM MANIKUMAR		2	2	2	1	2	2	1	0	1	2	8	9	8	6	10	12	56	60.0	60.0	60.0	75.0	65.0	Y	Y	Y	N	Y

98	19CS099	PONNAPATI DEEPAK KUMAR REDDY		2	2	2	1	2	2	2	2	2	2	2	2	2	2	52	80.0	80.0	80.0	80.0	80.0	Y	Y	Y	Y	Y
99	19CS100	PONNAPATI DINESH KUMAR REDDY		2	2	2	2	2	2	2	2	2	2	2	2	2	2	54	80.0	80.0	80.0	80.0	80.0	Y	Y	N	Y	Y
100	19CS101	POREDDY RANJITH KUMAR REDDY		2	2	2	2	2	2	2	2	2	2	2	2	2	2	73	90.0	90.0	90.0	90.0	85.0	Y	Y	Y	Y	Y
101	19CS102	POTHUPETA BHARGHAV		2	2	2	2	2	2	2	2	2	2	2	2	2	2	69	85.0	80.0	80.0	80.0	85.0	Y	Y	Y	Y	Y
102	19CS103	TELLA PRASANTH KUMAR		2	2	2	2	2	2	2	2	2	2	2	2	2	2	80	85.0	100.0	90.0	90.0	90.0	Y	Y	Y	Y	Y
103	19CS104	RACHURI LOKESH		2	2	2	2	2	2	2	2	2	2	2	2	2	2	74	90.0	80.0	80.0	90.0	85.0	Y	Y	Y	Y	Y
104	19CS105	RITHIKA V		2	2	2	2	2	2	2	2	2	2	2	2	2	2	78	90.0	90.0	85.0	85.0	90.0	Y	Y	Y	N	Y
105	19CS106	RUDRAPATI VISHNUVARDHAN		2	2	2	2	2	2	2	2	2	2	2	2	2	2	87	95.0	90.0	90.0	90.0	90.0	Y	Y	Y	Y	Y
106	19CS107	SAMUKTHA M		2	2	2	2	2	2	2	2	2	2	2	2	2	2	90	90.0	90.0	90.0	90.0	90.0	Y	Y	Y	Y	Y
107	19CS108	SANDHI BOINA SRINIVASULU		2	2	2	2	2	2	2	2	2	2	2	2	2	2	75	85.0	80.0	80.0	80.0	80.0	Y	Y	Y	Y	N
108	19CS109	SARAVANAN G		2	2	2	1	2	2	2	2	2	2	2	2	2	2	57	60.0	60.0	60.0	60.0	60.0	Y	Y	N	Y	N
109	19CS110	SELVAKUMAR R		2	2	2	2	2	2	2	2	2	2	2	2	2	2	78	85.0	90.0	90.0	85.0	90.0	Y	Y	Y	Y	Y
110	19CS111	SHANKAVI M		2	2	2	2	2	2	2	2	2	2	2	2	2	2	85	90.0	90.0	90.0	80.0	90.0	Y	Y	Y	Y	Y
111	19CS112	SIVAKUMAR S K		2	2	1	2	2	2	2	2	2	2	2	2	2	2	89	90.0	100.0	100.0	90.0	95.0	Y	Y	Y	Y	Y
112	19CS113	SOWNDHAR S		2	2	2	1	2	2	2	2	2	2	2	2	2	2	93	95.0	90.0	95.0	100.0	100.0	Y	Y	Y	Y	Y
113	19CS114	TELAGARA VISHNUKANTH REDDY		2	1	2	1	2	2	1	1	0	2	2	2	2	2	76	85.0	85.0	80.0	85.0	85.0	Y	Y	Y	Y	N
114	19CS115	THALISETTY KISHORE		2	2	1	2	2	2	2	2	0	2	2	2	2	2	63	45.0	90.0	50.0	45.0	90.0	N	Y	N	N	Y
115	19CS116	VALLURU PREETHIKA		2	2	1	2	2	2	2	2	1	2	2	2	2	2	87	95.0	85.0	80.0	85.0	90.0	Y	Y	Y	Y	Y
116	19CS117	VANDHAVASI YUVARAJU		2	2	2	2	2	2	2	2	0	2	2	2	2	2	74	70.0	80.0	85.0	80.0	65.0	Y	Y	Y	Y	Y
117	19CS118	VASANTHAPRIYA N J		2	2	2	2	0	2	2	2	2	2	2	2	2	2	90	100.0	90.0	80.0	95.0	100.0	Y	Y	Y	Y	Y
118	19CS119	VELPULA JAYA KRISHNA YADAV		2	2	2	2	2	1	2	0	2	2	2	2	2	2	57	95.0	60.0	50.0	50.0	90.0	Y	Y	Y	N	N
119	20CS501	AKASH A		2	2	2	2	2	2	1	2	2	2	2	2	2	2	62	70.0	85.0	75.0	80.0	90.0	Y	N	N	N	Y
120	20CS502	DHEVA M		2	2	0	2	2	1	2	2	2	2	2	2	2	2	90	100.0	85.0	90.0	100.0	95.0	Y	Y	Y	Y	Y
121	20CS503	DHIVYA R		2	2	2	2	2	2	2	2	2	2	2	2	2	2	80	85.0	85.0	85.0	85.0	80.0	N	Y	Y	Y	Y
122	20CS504	DINESH B		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	0.0	0.0	0.0	0.0	N	N	N	N	N
123	20CS505	HEMANTH E C		2	2	2	2	2	2	2	2	0	2	2	2	2	2	60	60.0	85.0	65.0	80.0	80.0	Y	N	Y	N	Y
124	20CS506	JAMUNA S		2	2	2	2	2	2	2	2	2	2	2	2	2	2	72	95.0	80.0	80.0	80.0	95.0	Y	Y	Y	N	Y
125	20CS507	KEERTHANA T		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	0.0	0.0	0.0	0.0	N	N	N	N	N
126	20CS508	MANIKANDAN M		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	0.0	0.0	0.0	0.0	N	N	N	N	N
127	20CS509	MUTHUVEL A		2	2	2	1	1	2	0	2	2	2	2	3	9	16	16	74	65.0	55.0	60.0	90.0	100.0	Y	N	Y	Y	Y
128	20CS510	PRAVEENKUMAR V		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	0.0	0.0	0.0	0.0	N	N	N	N	N
129	20CS511	RANJITH M		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	0.0	0.0	0.0	0.0	N	N	N	N	N
130	20CS512	SOUNDHARYA S		2	2	2	1	2	2	2	1	1	0	9	3	12	13	12	69	65.0	55.0	80.0	90.0	65.0	Y	N	Y	Y	Y
131	20CS513	SOUNDHAR S		2	2	2	2	0	0	1	2	0	1	9	5	8	6	5	45	65.0	45.0	40.0	45.0	30.0	Y	N	N	N	N

132	20CS514	VIMAL S		[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	67	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
133	20CS515	K.KIRITHISWARAN		[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	59	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
134	20CS516	S.TAMILSELVAN		[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	12	[illegible]	[illegible]	87	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

1. 2. Course Faculty

HOD 31/7/21

Regulation	R2016	Muthayammal Engineering College
Year/ Sem :	II/IV/A/B	Department of Computer science and Engineering
Batch	2019-2023	

Name of the Faculty: Mr.S.R.Sridhar & Mr.V.Karuppuchamy

Course Code & Name:19CSC12 & Software Engineering

Articulation Matrix

Course Outcomes	Program Outcomes (POs)												PSOs		
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
C313.CO1	3		3	3	3	2	-	-	3	-	3	2	-	-	3
C313.CO2	3	2	2	3	-	2	2	3	2	-	2	1	2	2	2
C313.CO3	3	2		3	3	2	-	-	3	2	-	-	-	3	-
C313.CO4	3	2		3	-	2	-	-	2	-	-	2	2	2	2
C313.CO5	3	3	2	2	-	-	-	-	3	1	-	2	1	3	2
Average	3	2.3	2.3	2.8	3	2	2	3	2.6	1.5	2.5	1.8	1.7	2.5	2.3

		Attainment Target
Attainment Level	3	More the 75 % of Students Attained 60 % of CO Target
Attainment Level	2	Between 60-75% of Students Attained 60 % of CO Target
Attainment Level	1	Between 50-60% of Students Attained 60 % of CO Target

Evaluation Pattern

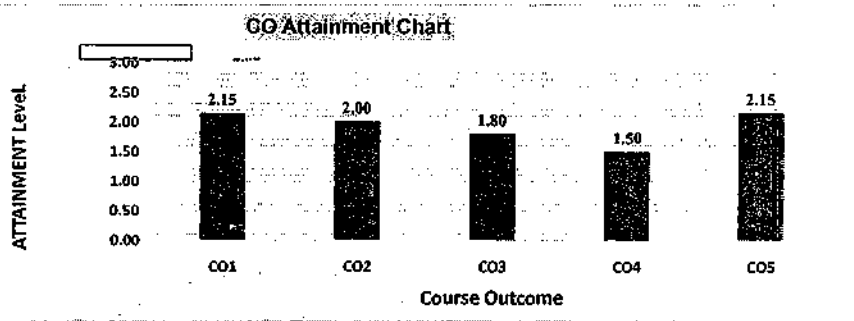
	CIA-I	CIA-II	CIA-III	Viva voce (MKC)	Assign	End Sem	Total Marks
Weightage	10	10	10	10	10	50	100
CO1	4.2		2	2		10	18.2
CO2	5.8		2	2		10	19.8
CO3		5.8	2	2		10	19.8
CO4		2.1	2	2		10	16.1
CO5			2	2	10	10	24

Assessment Pattern

	Mark	CIA-I	CIA-II	CIA-III	Viva voce (MKC)	Assign	End Sem
	Mark	50	50	100	10	10	100
CO1	21			20	2		20
CO2	29			20	2		20
CO3			29	20	2		20
CO4			21	20	2		20
CO5				20	2	10	20

Assessment Tools	CO1	CO2	CO3	CO4	CO5
Internal Assessment 1	3	2			
Internal Assessment 2			2	2	
Viva voce	3	3	2	3	3
Assignment					2
Model	1	1	1	1	2
End Semester Exam					

	CO1	CO2	CO3	CO4	CO5	Attainment
Internal Assessment CO Attainment	2.3	2	1.6	1	2.3	50%
End Semester Exam CO Attainment						
Overall CO Attainment						



Course POs and PSOs Attainment

Course Outcomes	Program Outcomes (POs)												PSOs		
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
C312.CO1	2.0	3.0	2.0	2.0		-	-	-	-	-	-	2.0	2.0	3.0	-
C312.CO2	2.0	2.0	2.0	3.0	3.0	-	-	2.0	-	2.0	3.0	-	2.0	2.0	-
C312.CO3	2.0	2.0	2.0	3.0		3.0	2.0	-	-	-	-	2.0	-	3.0	2.0
C312.CO4	3.0	2.0	-	3.0		-	-	-	-	-	2.0	2.0	3.0	2.0	-
C312.CO5	2.0	2.0	3.0	-	2.0	3.0	-	2.0	3.0	-	3.0	-	2.0	2.0	3.0
Average	2.2	2.2	2.3	2.8	2.5	3.0	2.0	2.0	3.0	2.0	2.7	2.0	2.3	2.4	2.5

Course Faculty

HOD

Interview Questions

SOFTWARE ENGINEERING Questions

SOFTWARE ENGINEERING Multiple Choice Questions :-

1. What are the characteristics of software?

a. Software is developed or engineered; it is not manufactured in the classical sense.
b. Software doesn't " wear out ".
c. Software can be custom built or custom build.
d. All mentioned above

ANSWER: All mentioned above

2. Compilers, Editors software come under which type of software?

a. System software
b. Application software
c. Scientific software
d. None of the above.

ANSWER: System software

3. Software Engineering is defined as systematic, disciplined and quantifiable approach for the development, operation and

maintenance of software.

a. True
b. False

ANSWER: True

4. RAD Software process model stands for _____ .

a. Rapid Application Development.
b. Relative Application Development.
c. Rapid Application Design.
d. Recent Application Development.

ANSWER: Rapid Application Development.

5. Software project management comprises of a number of activities, which contains _____.

a. Project planning
b. Scope management
c. Project estimation
d. All mentioned above

ANSWER: All mentioned above

6. COCOMO stands for _____ .

a. COnsumed COst MOdel
b. COnstructive COst MOdel
c. COmmon COntrol MOdel
d. COmposition COst MOdel

ANSWER: COnstructive COst MOdel

7. Which of the following is not defined in a good Software Requirement Specification (SRS) document?

a. Functional Requirement.
b. Nonfunctional Requirement.

c. Goals of implementation.
d. Algorithm for software implementation.

ANSWER: Algorithm for software implementation.

8. What is the simplest model of software development paradigm?

a. Spiral model
b. Big Bang model
c. V-model
d. Waterfall model

ANSWER: Waterfall model

9. Which of the following is the understanding of software product limitations, learning system related problems or changes to be done in existing systems beforehand, identifying and addressing the impact of project on organization and personnel etc?

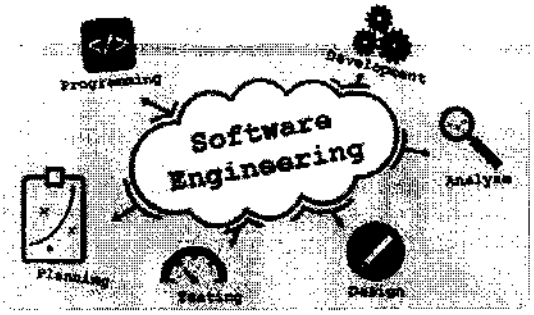
a. Software Design
b. Feasibility Study
c. Requirement Gathering
d. System Analysis

ANSWER: System Analysis

10. Which design identifies the software as a system with many components interacting with each other?

a. Architectural design
b. High-level design
c. Detailed design
d. Both B & C

ANSWER: Architectural design



SOFTWARE ENGINEERING Multiple Choice Questions

11. Software consists of _____ .

a. Set of instructions + operating procedures
b. Programs + documentation + operating procedures
c. Programs + hardware manuals
d. Set of programs

ANSWER: Programs + documentation + operating procedures

12. Which is the most important feature of spiral model?

a. Quality management
b. Risk management
c. Performance management
d. Efficiency management

ANSWER: Risk management

13. If every requirement stated in the Software Requirement Specification (SRS) has only one interpretation, SRS is said to be correct _____ .

a. Unambiguous
b. Consistent
c. Verifiable
d. None of the above

ANSWER: Unambiguous

14. Which is not a step of Requirement Engineering?

a. Requirements elicitation
b. Requirements analysis
c. Requirements design
d. Requirements documentation

ANSWER: Requirements design

15. FAST stands for _____ .

a. Functional Application Specification Technique
b. Fast Application Specification Technique
c. Facilitated Application Specification Technique
d. None of the above

ANSWER: Facilitated Application Specification Technique

16. The level at which the software uses scarce resources is _____ .

a. Reliability
b. Efficiency
c. Portability
d. All of the above

ANSWER: Efficiency

17. Modifying the software to match changes in the ever changing environment is called _____ .

a. Adaptive maintenance
b. Corrective maintenance
c. Perfective maintenance
d. Preventive maintenance

ANSWER: Adaptive maintenance

18. If every requirement can be checked by a cost-effective process, then the SRS is _____ .

17/2/2021
Subject faculty

HOD

X

Unit 3 - Week 2 :

Week 2 Assignment 2

The due date for submitting this assignment has passed.
As per our records you have not submitted this assignment. **Due on 2018-08-15, 23:59 IST.**

1) Which one of the following may be experienced by a software development team when it adopts a systematic development process model in preference to a build-and-fix style of development? ***1 point***

- a. Increased documentation overhead
- b. Increased development cost
- c. Decreased maintainability
- d. Increased development time

No, the answer is incorrect.
Score: 0

Accepted Answers:
a. Increased documentation overhead

2) A software process model represents which one of the following? ***1 point***

- a. The way in which software is developed
- b. The way in which software processes data
- c. The way in which software is used
- d. The way in which software may fail

No, the answer is incorrect.
Score: 0

Accepted Answers:
a. The way in which software is developed

3) Prototyping life cycle model is appropriate when a project suffers from which one of the following risks? ***1 point***

- a. Schedule slippage

Week 10 :
Week 11 :
Week 12 :
Download Videos
Assignment Solution
Interactive Session with Students - Software Engineering

Accepted Answers:
c. Incomplete and uncertain requirements

4) Which one of the following activities spans all stages of a software development life cycle (SDLC)? ***1 point***

- a. Coding
- b. Testing
- c. Project management
- d. Design

No, the answer is incorrect.
Score: 0

Accepted Answers:
c. Project management

5) The operation phase of the waterfall model is a synonym for which one of the following phases? ***1 point***

- a. Coding and unit testing phase
- b. Integration and system testing phase
- c. Maintenance phase
- d. Design phase

No, the answer is incorrect.
Score: 0

Accepted Answers:
c. Maintenance phase

6) The implementation phase of the waterfall model is a synonym for which one of the following phases? ***1 point***

- a. Coding and unit testing phase
- b. Integration and system testing phase
- c. Maintenance phase
- d. Design phase

No, the answer is incorrect.
Score: 0

Accepted Answers:
a. Coding and unit testing phase

7) Unit testing is carried out during which phase of the waterfall model? ***1 point***

- a. Implementation phase
- b. Testing phase
- c. Maintenance phase
- d. Design phase

No, the answer is incorrect.
Score: 0

Accepted Answers:
a. Implementation phase

8) Which one of the following phases accounts for the maximum effort during development of a typical software? ***1 point***

a. Coding

b. Testing

c. Designing

d. Specification

No, the answer is incorrect.
Score: 0

Accepted Answers:
b. Testing

9) Which one of the following is not a standard software development process model? ***1 point***

a. Waterfall Model

b. Recursive Model

c. RAD Model

d. V Model

No, the answer is incorrect.
Score: 0

Accepted Answers:
b. Recursive Model

10)Which one of the following feedback paths is not present in an iterative waterfall model? ***0 points***

a. Design phase to feasibility study phase

b. Implementation phase to design phase

c. Implementation phase to requirements specification phase

d. Design phase to requirements specification phase

No, the answer is incorrect.
Score: 0

Accepted Answers:
a. Design phase to feasibility study phase

Previous Page End

Subject faculty 17/2/2024

HOD

developer.*
The Independent Magazine for Software Professionals

Automating Software Development Processes

by Tim Kitchens

Automating repetitive procedures can provide real value to software development projects. In this article, we will explore the value of and barriers to automation and provide some guidance for automating aspects of the development process.

Although few experienced developers and project managers would argue the merits of automating development and testing procedures, when push comes to shove many teams place a low priority on implementing automated processes. The result is usually that, if automation is considered at all, it is given lip service early in the project life cycle, but falls quickly by the wayside.

Experience teaches us over and over again that trying to run a "simple" project by implementing a series of "simple" manual protocols, backed by "simple" written (sometimes, even just verbal) instructions, just doesn't work well. Even so, many of us still tend to allow ourselves to start the next project with the thought that the manual, protocol-based method will "do just fine."

After all, aren't we all professionals? Can't we read a set of simple instructions and just be disciplined enough to follow those instructions when the time is right? Isn't it a waste of time and money to invest in automating procedures for such a small project? The development team is only a half-dozen people after all—and the argument against automation goes on and on.

If you're a developer or tester who enjoys spending your time actually adding value to your project, rather than repeating the same routine tasks over and over, you'll want to consider advocating the concept of automation to your team (especially, to your project manager). If you're a project manager who's committed to maximizing the talents and time of the members of your technical team, as well as minimizing the risk of your project failing to deliver on time and on quality, you will want to encourage your team to invest the necessary time and effort required to automate the types of tasks that will be identified in this article.

Why Should I Automate?

You may already be familiar with many of the benefits of automating development processes. Some of the more commonly cited ones are:

Repeatability. Scripts can be repeated, and, unless your computer is having a particularly bad day, you can be reasonably certain that the same instructions will be executed in the same order each time the same script is run.

Reliability. Scripts reduce chances for human error.

Efficiency. Automated tasks will often be faster than the same task performed manually. (Some people might question whether gains in efficiency are typical, noting that they have worked on projects where, in their view, trying to automate tasks actually cost the project more time than it saved. Depending on the situation, this may be a real concern. In addition, automation might have been implemented poorly or carried too far on some projects—but keep reading for more on what to automate, and when.)

Testing. Scripted processes undergo testing throughout the development cycle, in much the same way the system code does. This greatly improves chances for successful process execution as the project progresses. Automated scripts eventually represent a mature, proven set of repeatable processes.

Versioning. Scripts are artifacts that can be placed under version control. With manual processes, the only artifacts that can be versioned and tracked are procedure documents. Versioning of human beings—the other factor in the manual process equation—is unfortunately not supported by your typical source control system.

Leverage. Another big benefit to automating is that developers and testers can focus on the areas where they add real value to a project—developing and testing new code and features—instead of worrying about the underlying development infrastructure issues.

For example, instead of requiring everyone to become intimately familiar with all the little nuances of the build procedure, you can have one person focus on automating the build and have that person provide the team with a greatly simplified method of building, hopefully as simple as running a command or two. Less time spent on builds leaves more time for the tasks that add the most value to the project.

What Should I Automate?

If you're convinced that automating your development processes is a good idea, the next logical question is which processes should be automated. While the answer, to some extent, is different for every project, there are some obvious ones, as well as some general guidelines that I'd like to offer. Some of the typical targets for automation are:

- Build and deployment of the system under design.
- Unit test execution and report generation.
- Code coverage report generation.
- Functional test execution and report generation.
- Load test execution and report generation.
- Code quality metrics report generation.
- Coding conventions report generation.

Deleted: s

The above list is obviously not exhaustive, and every project has its own unique characteristics. Here's a general, and perhaps obvious, rule of thumb to help identify any process that should be considered for automation: Consider automating processes that you expect to be repeated frequently throughout a system's life cycle. The more often the procedures will be repeated, the higher the value of automating them.

Once a process has been identified, spend a little time investigating how you might be able to automate the process, including researching tools that could assist with automation, and estimating the level of effort required to implement the automation versus the total cost and risk of requiring team members to manually perform the procedures. As with any other business decision, it really should come down to a cost versus benefit analysis.

You probably noticed that the term "report generation" appears in the above list of automation candidates. The repetition points out another important aspect of automating development processes: the end result of every automated process execution should be a report that is easily interpreted by the team. Ideally, such reports will focus on making anomalous conditions (for example, test failures) obvious at a glance. Also, these reports should be readily accessible to the appropriate team members.

Subject faculty 17/2/2024

HoD

Cornell University
Computing and Information Science

Software Engineering

Scenarios and Use Cases

Scenarios

Scenario

A **scenario** is a **scene** that illustrates some interaction with a proposed system.

A **scenario** is a tool used during requirements analysis to describe a specific use of a proposed system. Scenarios capture the system, as viewed from the outside, e.g., by a user, using specific examples.

Note on terminology

Some authors restrict the word "scenario" to refer to a user's total interaction with the system.

Other authors use the word "scenario" to refer to parts of the interaction.

In this course, the term is used with both meanings.

Describing a Scenario

Some organizations have complex documentation standards for describing a scenario.

At the very least, the description should include:

- A statement of the purpose of the scenario
- The individual user or transaction that is being followed through the scenario
- Assumptions about equipment or software
- The steps of the scenario

Developing a Scenario with a Client

Example of how to develop a scenario with a client

The requirements are being developed for a system that will enable university students to take exams online from their own rooms using a web browser.

Create a scenario for how a typical student interacts with the system.

In the next few slides, the questions in blue are typical of the questions to ask the client while developing the scenario.

Developing a Scenario with a Client: a Typical Student

Purpose: Scenario that describes the use of an online Exam system by a representative student

Individual: *[Who is a typical student?]* Student A, senior at Cornell, major in computer science. *[Where can the student be located? Do other universities differ?]*

Equipment: Any computer with a supported browser. *[Is there a list of supported browsers? Are there any network restrictions?]*

Scenario:

1. Student A authenticates. *[How does a Cornell student authenticate?]*
2. Student A starts browser and types URL of Exam system. *[How does the student know the URL?]*
3. Exam system displays list of options. *[Is the list tailored to the individual user?]*

17/2/2024
Subject faculty

HoD

Proceedings of the 25th International Conference on Software Engineering, IEEE Computer Society, 2003, pp. 726-736.

Writing Good Software Engineering Research Papers

Minitutorial

Mary Shaw
Carnegie Mellon University
mary.shaw@cs.cmu.edu

Abstract

Software engineering researchers solve problems of several different kinds. To do so, they produce several different kinds of results, and they should develop appropriate evidence to validate these results. They often report their research in conference papers. I analyzed the abstracts of research papers submitted to ICSE 2002 in order to identify the types of research reported in the submitted and accepted papers, and I observed the program committee discussions about which papers to accept. This report presents the research paradigms of the papers, common concerns of the program committee, and statistics on success rates. This information should help researchers design better research projects and write papers that present their results to best advantage.

Keywords: research design, research paradigms, validation, software profession, technical writing

1. Introduction

In software engineering, research papers are customary vehicles for reporting results to the research community. In a research paper, the author explains to an interested reader what he or she accomplished, and how the author accomplished it, and why the reader should care. A good research paper should answer a number of questions:

- What, precisely, was your contribution?
  - What question did you answer?
  - Why should the reader care?
  - What larger question does this address?
- What is your new result?
  - What new knowledge have you contributed that the reader can use elsewhere?
  - What previous work (yours or someone else's) do you build on? What do you provide a superior alternative to?
  - How is your result different from and better than this prior work?
  - What, precisely and in detail, is your new result?
- Why should the reader believe your result?
  - What standard should be used to evaluate your claim?
  - What concrete evidence shows that your result satisfies your claim?

If you answer these questions clearly, you'll probably communicate your result well. If in addition your result represents an interesting, sound, and significant contribution to our knowledge of software engineering, you'll have a good chance of getting it accepted for publication in a conference or journal.

Other fields of science and engineering have well-established research paradigms. For example, the experimental model of physics and the double-blind studies of medicines are understood, at least in broad outline, not only by the research community but also by the public at large. In addition to providing guidance for the design of research in a discipline, these paradigms establish the scope of scientific disciplines through a social and political process of "boundary setting" [5].

Software engineering, however, has not yet developed this sort of well-understood guidance. I previously [19, 20] discussed early steps toward such understanding, including a model of the way software engineering techniques mature [17, 18] and critiques of the lack of rigor in experimental software engineering [1, 22, 23, 24, 25]. Those discussions critique software engineering research reports against the standards of classical paradigms. The discussion here differs from those in that this discussion reports on the types of papers that are accepted in practices as good research reports. Another current activity, the Impact Project [7] seeks to trace the influence of software engineering research on practice. The discussion here focuses on the paradigms rather than the content of the research

This report examines how software engineers answer the questions above, with emphasis on the design of the research project and the organization of the report. Other sources (e.g., [4]) deal with specific issues of technical writing. Very concretely, the examples here come from the papers submitted to ICSE 2002 and the program committee review of those papers. These examples report research results in software engineering. Conferences often include other kinds of papers, including experience reports, materials on software engineering education, and opinion essays.

Table 2 gives the distribution of submissions to ICSE 2002, based on reading the abstracts (not the full papers—but remember that the abstract tells a reader what to expect from the paper). For each type of research question, the table gives the number of papers submitted and accepted, the percentage of the total paper set of each kind, and the acceptance ratio within each type of question. Figures 1 and 2 show these counts and distributions.

Table 2. Types of research questions represented in ICSE 2002 submissions and acceptances

Type of question	Submitted	Accepted	Ratio Acc/Sub
Method or means of development	142(48%)	18 (42%)	(13%)
Method for analysis or evaluation	95 (32%)	19 (44%)	(20%)
Design, evaluation, or analysis of a particular instance	43 (14%)	5 (12%)	(12%)
Generalization or characterization	18 (6%)	1 (2%)	(6%)
Feasibility study or exploration	0 (0%)	0 (0 %)	(0%)
TOTAL	298(100.0%)	43 (100.0%)	(14%)

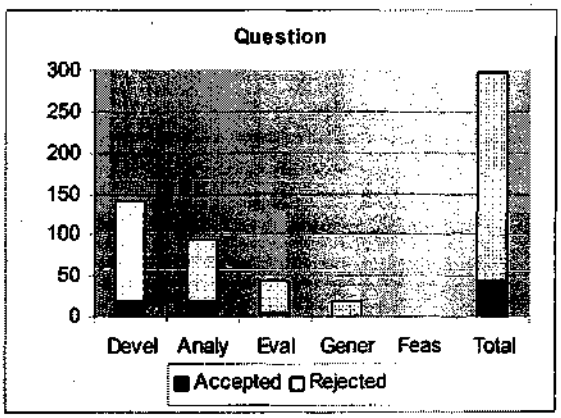


Figure 1. Counts of acceptances and rejections by type of research question

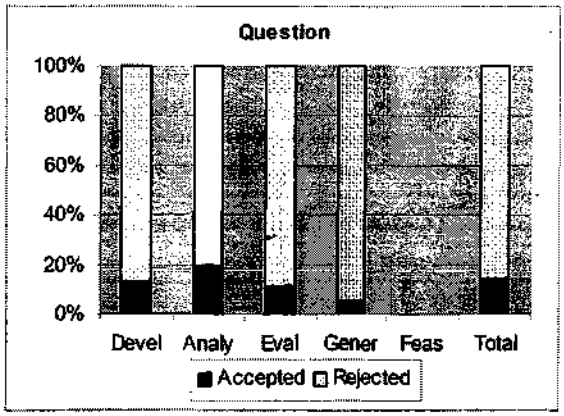


Figure 2. Distribution of acceptances and rejections by type of research question

2.3 What do program committees look for?

Acting on behalf of prospective readers, the program committee looks for a clear statement of the specific problem you solved—the question about software development you answered—and an explanation of how the answer will help solve an important software engineering problem. You'll devote most of your paper to describing your result, but you should begin by explaining what question you're answering and why the answer matters.

If the program committee has trouble figuring out whether you developed a new evaluation technique and demonstrated it on an example, or applied a technique you reported last year to a new real-world example, or evaluated the use of a well-established evaluation technique, you have not been clear.

3. What is your new result?

Explain precisely what you have contributed to the store of software engineering knowledge and how this is useful beyond your own project.

3.1 What kinds of results do software engineers produce?

The tangible contributions of software engineering research may be procedures or techniques for development or analysis; they may be models that generalize from specific examples, or they may be specific tools, solutions, or results about particular systems. Table 3 lists the types of research results that are reported in software engineering research papers and provides specific examples.

3.2 Which of these are most common?

By far the most common kind of ICSE paper reports a new procedure or technique for development or analysis. Models of various degrees of precision and formality were also common, with better success rates for quantitative than for qualitative models. Tools and notations were well represented, usually as auxiliary results in combination with a procedure or technique. Table 4 gives the distribution of submissions to ICSE 2002, based on reading the abstracts (but not the papers), followed by graphs of the counts and distributions in Figures 3 and 4.

The number of results is larger than the number of papers because 50 papers included a supporting result, usually a tool or a qualitative model.

Research projects commonly produce results of several kinds. However, conferences, including ICSE, usually impose strict page limits. In most cases, this provides too little space to allow full development of more than one idea, perhaps with one or two supporting ideas. Many authors present the individual ideas in conference papers, and then synthesize them in a journal article that allows space to develop more complex relations among results.

3.3 What do program committees look for?

The program committee looks for interesting, novel, exciting results that significantly enhance our ability to develop and maintain software, to know the quality of the software we develop, to recognize general principles about software, or to analyze properties of software.

You should explain your result in such a way that someone else could use your ideas. Be sure to explain what's novel or original – is it the idea, the application of the idea, the implementation, the analysis, or what?

Define critical terms precisely. Use them consistently. The more formal or analytic the paper, the more important this is.

Here are some questions that the program committee may ask about your paper:

What, precisely, do you claim to contribute?

Does your result fully satisfy your claims? Are the definitions precise, and are terms used consistently?

Authors tend to have trouble in some specific situations. Here are some examples, with advice for staying out of trouble:

- *If your result ought to work on large systems*, explain why you believe it scales.
- *If you claim your method is "automatic"*, using it should not require human intervention. If it's automatic when it's operating but requires manual assistance to configure, say so. If it's automatic except for certain cases, say so, and say how often the exceptions occur.
- *If you claim your result is "distributed"*, it probably should not have a single central controller or server. If it does, explain what part of it is distributed and what part is not.
- *If you're proposing a new notation for an old problem*, explain why your notation is clearly superior to the old one.
- *If your paper is an "experience report"*, relating the use of a previously-reported tool or technique in a practical software project, be sure that you explain what idea the reader can take away from the paper to use in other settings. If that idea is increased confidence in the tool or technique, show how your experience should increase the reader's confidence for applications beyond the example of the paper.

What's new here?

The program committee wants to know what is novel or exciting, and why. What, specifically, is the contribution? What is the increment over earlier work by the same authors? by other authors? Is this a sufficient increment, given the usual standards of subdiscipline?

Above all, the program committee also wants to know what you actually contributed to our store of knowledge about software engineering. Sure, you wrote this tool and tried it out. But was your contribution the technique that is embedded in the tool, or was it making a tool that's more effective than other tools that implement the technique, or was it showing that the tool you described in a previous paper actually worked on a practical large-scale problem? It's better for you as the author to explain than for the program committee to guess. Be clear about your claim ...

Awful	▼	• I completely and generally solved ... (unless you actually did!)
Bad	▼	• I worked on galumphing. (or studied, investigated, sought, explored)
Poor	▼	• I worked on improving galumphing. (or contributed to, participated in, helped with)
Good	▲	• I showed the feasibility of composing blitzing with flitzing. • I significantly improved the accuracy of the standard detector. (or proved, demonstrated, created, established, found, developed)
Better	▲	• I automated the production of flitz tables from specifications. • With a novel application of the blivet transform, I achieved a 10% increase in speed and a 15% improvement in coverage over the standard method.

Use verbs that show results and achievement, not just effort and activity.

> "***Try*** not. ***Do,*** or ***do*** not. There is no ***try***." -- Yoda

What has been done before? How is your work different or better?

What existing technology does your research build on? What existing technology or prior research does your research provide a superior alternative to? What's new here compared to your own previous work? What alternatives have other researchers pursued, and how is your work different or better?

4. Why should the reader believe your result?

Show evidence that your result is valid—that it actually helps to solve the problem you set out to solve.

4.1. What kinds of validation do software engineers do?

Software engineers offer several kinds of evidence in support of their research results. It is essential to select a form of validation that is appropriate for the type of research result and the method used to obtain the result. As an obvious example, a formal model should be supported by rigorous derivation and proof, not by one or two simple examples. On the other hand, a simple example derived from a practical system may play a major role in validating a new type of development method. Table 5 lists the types of research validation that are used in software engineering research papers and provides specific examples. In this table, the examples are keyed to the type of result they apply to.

Table 5. Types of software engineering research validation

Type of validation	Examples
Analysis	I have analyzed my result and find it satisfactory through rigorous analysis, e.g. ... For a formal model ... rigorous derivation and proof For an empirical model ... data on use in controlled situation For a controlled experiment ... carefully designed experiment with statistically significant results
Evaluation	Given the stated criteria, my result... For a descriptive model ... adequately describes phenomena of interest ... For a qualitative model ... accounts for the phenomena of interest... For an empirical model ... is able to predict ... because ..., or ... generates results that fit actual data ... Includes feasibility studies, pilot projects
Experience	My result has been used on real examples by someone other than me, and the evidence of its correctness/usefulness/effectiveness is ... For a qualitative model ... narrative For an empirical model or tool ... data, usually statistical, on practice For a notation or technique ... comparison of systems in actual use
Example	Here's an example of how it works on For a technique or procedure ...a "slice of life" example based on a real system ... For a technique or procedure ...a system that I have been developing ... For a technique or procedure ... a toy example, perhaps motivated by reality The "slice of life" example is most likely to be convincing, especially if accompanied by an explanation of why the simplified example retains the essence of the problem being solved. Toy or textbook examples often fail to provide persuasive validation, (except for standard examples used as model problems by the field).
Persuasion	I thought hard about this, and I believe passionately that ... For a technique ... if you do it the following way, then ... For a system ... a system constructed like this would ... For a model ... this example shows how my idea works Validation purely by persuasion is rarely sufficient for a research paper. Note, though, that if the original question was about feasibility, a working system, even without analysis, can suffice
Blatant assertion	No serious attempt to evaluate result. This is highly unlikely to be acceptable

4.2 Which of these are most common?

Alas, well over a quarter of the ICSE 2002 abstracts give no indication of how the paper's results are validated, if at all. Even when the abstract mentions that the result was applied to an example, it was not always clear whether the example was a textbook example, or a report on use in the field, or something in between.

The most successful kinds of validation were based on analysis and real-world experience. Well-chosen examples were also successful. Persuasion was not persuasive, and narrative evaluation was only slightly more successful. Table 6 gives the distribution of submissions to ICSE 2002, based on reading the abstracts (but not the papers), followed by graphs of the counts and distributions. Figures 5 and 6 show these counts and distributions.

- *If you performed a controlled experiment,* explain the experimental design. What is the hypothesis? What is the treatment? What is being controlled? What data did you collect, and how did you analyze it? Are the results significant? What are the potentially confounding factors, and how are they handled? Do the conclusions follow rigorously from the experimental data?
- *If you performed an empirical study,* explain what you measured, how you analyzed it, and what you concluded. What data did you collect, and how? How is the analysis related to the goal of supporting your claim about the result? Do not confuse correlation with causality.
- *If you use a small example for explaining the result,* provide additional evidence of its practical use and scalability.

5. How do you combine the elements into a research strategy?

It is clear that not all combinations of a research question, a result, and a validation strategy lead to good research. Software engineering has not developed good general guidance on this question.

Tables 1, 3, and 5 define a 3-dimensional space. Some portions of that space are densely populated: One common paradigm is to find a better way to perform some software development or maintenance task, realize this in a concrete procedure supported by a tool, and evaluate the effectiveness of this procedure and tool by determining how its use affects some measure (e.g., error rates) of quality. Another common paradigm is to find a better way to evaluate a formalizable property of a software system, develop a formal model that supports inference, and to show that the new model allows formal analysis or proof of the properties of interest.

Clearly, the researcher does not have free choice to mix and match the techniques—validating the correctness of a formal model through field study is as inappropriate as attempting formal verification of a method based on good organization of rules of thumb.

Selecting a type of result that will answer a given question usually does not seem to present much difficulty, at least for researchers who think carefully about the choice. Blindly adopting the research paradigm someone used last year for a completely different problem is a different case, of course, and it can lead to serious misfits.

Choosing a good form of validation is much harder, and this is often a source of difficulty in completing a successful paper. Table 6 shows some common good matches. This does not, unfortunately, provide complete guidance.

When I advise PhD students on the validation section of their theses, I offer the following heuristic: Look carefully at the short statement of the result—the principal claim of the thesis. This often has two or three clauses (e.g., I found an efficient and complete method ..."); if so, each presents a separate validation problem. Ask of each clause whether it is a global statement ("always", "fully"), a qualified statement ("a 25% improvement", "for noncyclic structures..."), or an existential statement ("we found an instance of"). Global statements often require analytic validation, qualified statements can often be validated by evaluation or careful examination of experience, and existential statements can sometimes be validated by a single positive example. A frequent result of this discussion is that students restate the thesis claims to reflect more precisely what the theses actually achieve. If we have this discussion early enough in the thesis process, students think about planning the research with demonstrable claims in mind.

Concretely, Table 7 shows the combinations that were represented among the accepted papers at ICSE 2002, omitting the 7 for which the abstracts were unclear about validation:

Table 7. Paradigms of ICSE2002 acceptances

Question	Result	Validation	#
Devel method	Procedure	Analysis	2
Devel method	Procedure	Experience	3
Devel method	Procedure	Example	3
Devel method	Qual model	Experience	2
Devel method	Analytic model	Experience	2
Devel method	Notation or tool	Experience	1
Analysis method	Procedure	Analysis	5
Analysis method	Procedure	Evaluation	1
Analysis method	Procedure	Experience	2
Analysis method	Procedure	Example	6
Analysis method	Analytic model	Experience	1
Analysis method	Analytic model	Example	2
Analysis method	Tool	Analysis	1
Eval of instance	Specific analysis	Analysis	3
Eval of instance	Specific analysis	Example	2

6. Does the abstract matter?

The abstracts of papers submitted to ICSE convey a sense of the kinds of research submitted to the conference. Some abstracts were easier to read and (apparently) more informative than others. Many of the clearest abstracts had a common structure:

- Two or three sentences about the current state of the art, identifying a particular problem
- One or two sentences about what this paper contributes to improving the situation

which would not have been possible without the cooperation and encouragement of the ICSE 2002 program committee. The development of these ideas has also benefited from discussion with the ICSE 2002 program committee, with colleagues at Carnegie Mellon, and at open discussion sessions at FSE Conferences. The work has been supported by the A. J. Perlis Chair at Carnegie Mellon University.

9. References

1. Victor R. Basili. The experimental paradigm in software engineering. In *Experimental Software Engineering Issues: Critical Assessment and Future Directives*. Proc of Dagstuhl-Workshop, H. Dieter Rombach, Victor R. Basili, and Richard Selby (eds), published as *Lecture Notes in Computer Science #706*, Springer-Verlag 1993.
2. Geoffrey Bowker and Susan Leigh Star: *Sorting Things Out: Classification and Its Consequences*. MIT Press, 1999
3. Frederick P. Brooks, Jr. Grasping Reality Through Illusion—Interactive Graphics Serving Science. *Proc 1988 ACM SIGCHI Human Factors in Computer Systems Conf (CHI '88)* pp. 1-11.
4. Rebecca Burnett. *Technical Communication*. Thomson Heinle 2001.
5. Thomas F. Gieryn. *Cultural Boundaries of Science: Credibility on the line*. Univ of Chicago Press, 1999.
6. ICSE 2002 Program Committee. *Types of ICSE papers*. http://icse-conferences.org/2002/info/paperTypes.html
7. Impact Project. "Determining the impact of software' engineering research upon practice. Panel summary, *Proc. 23rd International Conference on Software Engineering (ICSE 2001)*, 2001
8. Ellen Isaacs and John Tang. *Why don't more non-North-American papers get accepted to CHI?* http://acm.org/sigchi/bulletin/1996.1/isaacs.html
9. Ralph E. Johnson & panel. How to Get a Paper Accepted at OOPSLA. *Proc OOPSLA'93*, pp. 429-436, http://acm.org/sigplan/oopsla/oopsla96/how93.html
10. Jim Kajiya. How to Get Your SIGGRAPH Paper Rejected. Mirrored at http://www.cc.gatech.edu/student.services/phd/phd-advice/kajiya
11. Roy Levin and David D. Redell. How (and How Not) to Write a Good Systems Paper. *ACM SIGOPS Operating Systems Review*, Vol. 17, No. 3 (July, 1983), pages 35-40. http://ftp.digital.com/pub/DEC/SRC/other/SOSPadvice.txt
12. William Newman. A preliminary analysis of the products of HCI research, using pro forma abstracts. *Proc 1994 ACM SIGCHI Human Factors in Computer Systems Conf (CHI '94)*, pp.278-284.
13. William Newman et al. *Guide to Successful Papers Submission at CHI 2001*. http://acm.org/sigs/sigchi/chi2001/call/submissions/guide-papers.html
14. OOPSLA '91 Program Committee. How to get your paper accepted at OOPSLA. *Proc OOPSLA'91*, pp.359-363. http://acm.org/sigplan/oopsla/oopsla96/how91.html
15. Craig Partridge. How to Increase the Chances your Paper is Accepted at ACM SIGCOMM. http://www.acm.org/sigcomm/conference-misc/author-guide.html
16. William Pugh and PDLI 1991 Program Committee. *Advice to Authors of Extended Abstracts*. http://acm.org/sigsoft/conferences/pughadvice.html
17. Samuel Redwine, et al. *DoD Related Software Technology Requirements, Practices, and Prospects for the Future*. IDA Paper P-1788, June 1984.
18. S. Redwine & W. Riddle. Software technology maturation. *Proceedings of the Eighth International Conference on Software Engineering*, May 1985, pp. . 189-200.
19. Mary Shaw. The coming-of-age of software architecture research. *Proc. 23rd Int'l Conf on Software Engineering (ICSE 2001)*, pp. 656-664a.
20. Mary Shaw. What makes good research in software engineering? Presented at ETAPS 02, appeared in Opinion Corner department, *Int'l Jour on Software Tools for Tech Transfer*, vol 4, DOI 10.1007/s10009-002-0083-4, June 2002.
21. SigGraph 2003 Call for Papers. http://www.siggraph.org/s2003/cfp/papers/index.html
22. W. F. Tichy, P. Lukowicz, L. Prechelt, & E. A. Heinz. "Experimental evaluation in computer science: A quantitative study." *Journal of Systems Software*, Vol. 28, No. 1, 1995, pp. 9-18.
23. Walter F. Tichy. "Should computer scientists experiment more? 16 reasons to avoid experimentation." *IEEE Computer*, Vol. 31, No. 5, May 1998
24. Marvin V. Zelkowitz and Delores Wallace. Experimental validation in software engineering. *Information and Software Technology*, Vol 39, no 11, 1997, pp. 735-744.
25. Marvin V. Zelkowitz and Delores Wallace. Experimental models for validating technology. *IEEE Computer*, Vol. 31, No. 5, 1998, pp.23-31.
26. Mary-Claire van Leunen and Richard Lipton. *How to have your abstract rejected.* http://acm.org/sigsoft/conferences/vanLeunenLipton.html

17/2/2004
Subject faculty

HOD

Proceedings of the 25th International Conference on Software Engineering, IEEE Computer Society, 2003. pp. 726-736.

Writing Good Software Engineering Research Papers

Minitutorial

Mary Shaw
Carnegie Mellon University
mary.shaw@cs.cmu.edu

Abstract

Software engineering researchers solve problems of several different kinds. To do so, they produce several different kinds of results, and they should develop appropriate evidence to validate these results. They often report their research in conference papers. I analyzed the abstracts of research papers submitted to ICSE 2002 in order to identify the types of research reported in the submitted and accepted papers, and I observed the program committee discussions about which papers to accept. This report presents the research paradigms of the papers, common concerns of the program committee, and statistics on success rates. This information should help researchers design better research projects and write papers that present their results to best advantage.

Keywords: research design, research paradigms, validation, software profession, technical writing

1. Introduction

In software engineering, research papers are customary vehicles for reporting results to the research community. In a research paper, the author explains to an interested reader what he or she accomplished, and how the author accomplished it, and why the reader should care. A good research paper should answer a number of questions:

- What, precisely, was your contribution?
  - What question did you answer?
  - Why should the reader care?
  - What larger question does this address?
- What is your new result?
  - What new knowledge have you contributed that the reader can use elsewhere?
  - What previous work (yours or someone else's) do you build on? What do you provide a superior alternative to?
  - How is your result different from and better than this prior work?
  - What, precisely and in detail, is your new result?
- Why should the reader believe your result?
  - What standard should be used to evaluate your claim?
  - What concrete evidence shows that your result satisfies your claim?

If you answer these questions clearly, you'll probably communicate your result well. If in addition your result represents an interesting, sound, and significant contribution to our knowledge of software engineering, you'll have a good chance of getting it accepted for publication in a conference or journal.

Other fields of science and engineering have well-established research paradigms. For example, the experimental model of physics and the double-blind studies of medicines are understood, at least in broad outline, not only by the research community but also by the public at large. In addition to providing guidance for the design of research in a discipline, these paradigms establish the scope of scientific disciplines through a social and political process of "boundary setting" [5].

Software engineering, however, has not yet developed this sort of well-understood guidance. I previously [19, 20] discussed early steps toward such understanding, including a model of the way software engineering techniques mature [17, 18] and critiques of the lack of rigor in experimental software engineering [1, 22, 23, 24, 25]. Those discussions critique software engineering research reports against the standards of classical paradigms. The discussion here differs from those in that this discussion reports on the types of papers that are accepted in practices as good research reports. Another current activity, the Impact Project [7] seeks to trace the influence of software engineering research on practice. The discussion here focuses on the paradigms rather than the content of the research

This report examines how software engineers answer the questions above, with emphasis on the design of the research project and the organization of the report. Other sources (e.g., [4]) deal with specific issues of technical writing. Very concretely, the examples here come from the papers submitted to ICSE 2002 and the program committee review of those papers. These examples report research results in software engineering. Conferences often include other kinds of papers, including experience reports, materials on software engineering education, and opinion essays.

2. What, precisely, was your contribution?

Before reporting what you did, explain what problem you set out to solve or what question you set out to answer —and why this is important.

2.1 What kinds of questions do software engineers investigate?

Generally speaking, software engineering researchers seek better ways to develop and evaluate software. Development includes all the synthetic activities that involve creating and modifying the software, including the code, design documents, documentation, etc. Evaluation includes all the analytic activities associated with predicting, determining, and estimating properties of the software systems, including both functionality and extra-functional properties such as performance or reliability.

Software engineering research answers questions about methods of development or analysis, about details of designing or evaluating a particular instance, about generalizations over whole classes of systems or techniques, or about exploratory issues concerning existence or feasibility. Table 1 lists the types of research questions that are asked by software engineering research papers and provides specific question templates.

Table 1. Types of software engineering research questions

Type of question	Examples
Method or means of development	How can we do/create/modify/evolve (or automate doing) X? What is a better way to do/create/modify/evolve X?
Method for analysis or evaluation	How can I evaluate the quality/correctness of X? How do I choose between X and Y?
Design, evaluation, or analysis of a particular instance	How good is Y? What is property X of artifact/method Y? What is a (better) design, implementation, maintenance, or adaptation for application X? How does X compare to Y? What is the current state of X / practice of Y?
Generalization or characterization	Given X, what will Y (necessarily) be? What, exactly, do we mean by X? What are its important characteristics? What is a good formal/empirical model for X? What are the varieties of X, how are they related?
Feasibility study or exploration	Does X even exist, and if so what is it like? Is it possible to accomplish X at all?

The first two types of research produce methods of development or of analysis that the authors investigated in one setting, but that can presumably be applied in other settings. The third type of research deals explicitly with some particular system, practice, design or other instance of a system or method; these may range from narratives about industrial practice to analytic comparisons of alternative designs. For this type of research the instance itself should have some broad appeal—an evaluation of Java is more likely to be accepted than a simple evaluation of the toy language you developed last summer. Generalizations or characterizations explicitly rise above the examples presented in the paper. Finally, papers that deal with an issue in a completely new way are sometimes treated differently from papers that improve on prior art, so "feasibility" is a separate category (though no such papers were submitted to ICSE 2002).

Newman's critical comparison of HCI and traditional engineering papers [12] found that the engineering papers were mostly incremental (improved model, improved technique), whereas many of the HCI papers broke new ground (observations preliminary to a model, brand new technique). One reasonable interpretation is that the traditional engineering disciplines are much more mature than HCI, and so the character of the research might reasonably differ [17, 18]. Also, it appears that different disciplines have different expectations about the "size" of a research result—the extent to which it builds on existing knowledge or opens new questions. In the case of ICSE, the kinds of questions that are of interest and the minimum interesting increment may differ from one area to another.

2.2 Which of these are most common?

The most common kind of ICSE paper reports an improved method or means of developing software—that is, of designing, implementing, evolving, maintaining, or otherwise operating on the software system itself. Papers addressing these questions dominate both the submitted and the accepted papers. Also fairly common are papers about methods for reasoning about software systems, principally analysis of correctness (testing and verification). Analysis papers have a modest acceptance edge in this very selective conference.

Table 2 gives the distribution of submissions to ICSE 2002, based on reading the abstracts (not the full papers—but remember that the abstract tells a reader what to expect from the paper). For each type of research question, the table gives the number of papers submitted and accepted, the percentage of the total paper set of each kind, and the acceptance ratio within each type of question. Figures 1 and 2 show these counts and distributions.

Table 2. Types of research questions represented in ICSE 2002 submissions and acceptances

Type of question	Submitted	Accepted	Ratio Acc/Sub
Method or means of development	142(48%)	18(42%)	(13%)
Method for analysis or evaluation	95(32%)	19(44%)	(20%)
Design, evaluation, or analysis of a particular instance	43(14%)	5 (12%)	(12%)
Generalization or characterization	18(6%)	1 (2%)	(6%)
Feasibility study or exploration	0 (0%)	0 (0%)	(0%)
TOTAL	298(100.0%)	43(100.0%)	(14%)

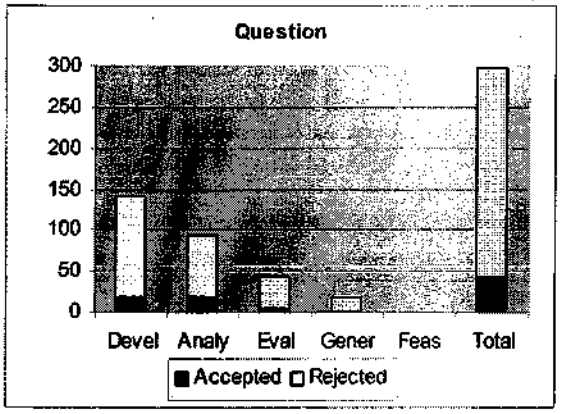


Figure 1. Counts of acceptances and rejections by type of research question

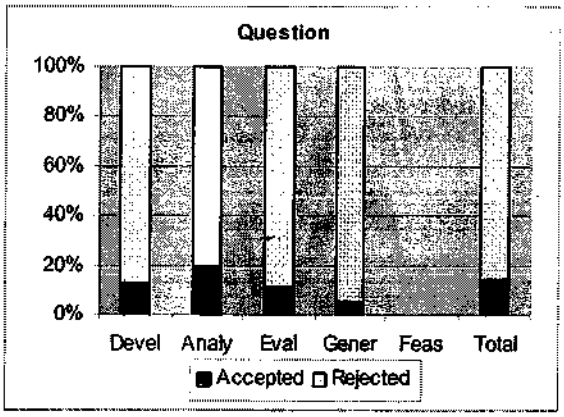


Figure 2. Distribution of acceptances and rejections by type of research question

2.3 What do program committees look for?

Acting on behalf of prospective readers, the program committee looks for a clear statement of the specific problem you solved—the question about software development you answered—and an explanation of how the answer will help solve an important software engineering problem. You'll devote most of your paper to describing your result, but you should begin by explaining what question you're answering and why the answer matters.

If the program committee has trouble figuring out whether you developed a new evaluation technique and demonstrated it on an example, or applied a technique you reported last year to a new real-world example, or evaluated the use of a well-established evaluation technique, you have not been clear.

3. What is your new result?

Explain precisely what you have contributed to the store of software engineering knowledge and how this is useful beyond your own project.

3.1 What kinds of results do software engineers produce?

The tangible contributions of software engineering research may be procedures or techniques for development or analysis; they may be models that generalize from specific examples, or they may be specific tools, solutions, or results about particular systems. Table 3 lists the types of research results that are reported in software engineering research papers and provides specific examples.

3.2 Which of these are most common?

By far the most common kind of ICSE paper reports a new procedure or technique for development or analysis. Models of various degrees of precision and formality were also common, with better success rates for quantitative than for qualitative models. Tools and notations were well represented, usually as auxiliary results in combination with a procedure or technique. Table 4 gives the distribution of submissions to ICSE 2002, based on reading the abstracts (but not the papers), followed by graphs of the counts and distributions in Figures 3 and 4.

Table 3. Types of software engineering research results	
Type of result	**Examples**
Procedure or technique	New or better way to do some task, such as design, implementation, maintenance, measurement, evaluation, selection from alternatives; includes techniques for implementation, representation, management, and analysis; a technique should be operational—not advice or guidelines, but a procedure
Qualitative or descriptive model	Structure or taxonomy for a problem area; architectural style, framework, or design pattern; non-formal domain analysis, well-grounded checklists, well-argued informal generalizations, guidance for integrating other results, well-organized interesting observations
Empirical model	Empirical predictive model based on observed data
Analytic model	Structural model that permits formal analysis or automatic manipulation
Tool or notation	Implemented tool that embodies a technique; formal language to support a technique or model (should have a calculus, semantics, or other basis for computing or doing inference)
Specific solution, prototype, answer, or judgment	Solution to application problem that shows application of SE principles – may be design, prototype, or full implementation; careful analysis of a system or its development, result of a specific analysis, evaluation, or comparison
Report	Interesting observations, rules of thumb, but not sufficiently general or systematic to rise to the level of a descriptive model.

Table 4. Types of research results represented in ICSE 2002 submissions and acceptances			
Type of result	**Submitted**	**Accepted**	**Ratio Acc/Sub**
Procedure or technique	152(44%)	28(51%)	18%
Qualitative or descriptive model	50(14%)	4 (7%)	8%
Empirical model	4 (1%)	1 (2%)	25%
Analytic model	48(14%)	7 (13%)	15%
Tool or notation	49(14%)	10(18%)	20%
Specific solution, prototype, answer, or judgment	34(10%)	5 (9%)	15%
Report	11(3%)	0 (0%)	0%
TOTAL	348(100.0%)	55(100.0%)	16%

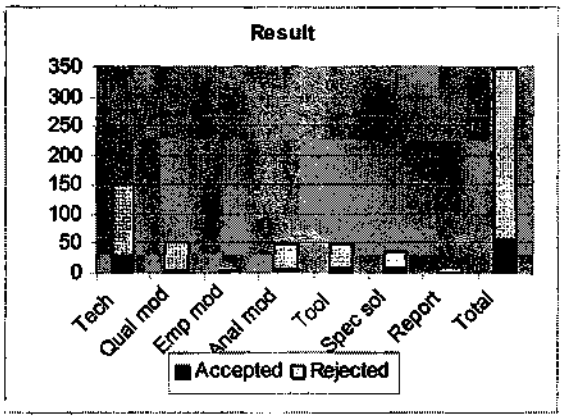


Figure 3. Counts of acceptances and rejections by type of result

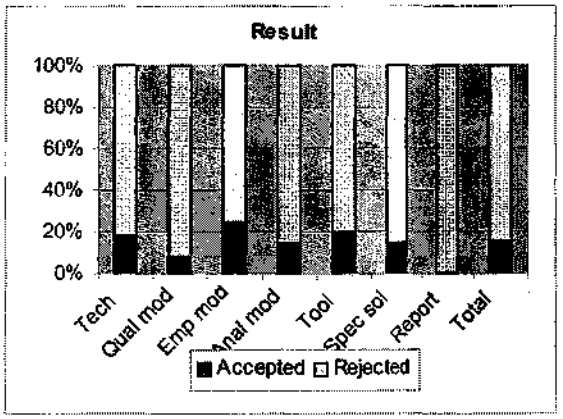


Figure 4. Distribution of acceptances and rejections by type of result

The number of results is larger than the number of papers because 50 papers included a supporting result, usually a tool or a qualitative model.

Research projects commonly produce results of several kinds. However, conferences, including ICSE, usually impose strict page limits. In most cases, this provides too little space to allow full development of more than one idea, perhaps with one or two supporting ideas. Many authors present the individual ideas in conference papers, and then synthesize them in a journal article that allows space to develop more complex relations among results.

3.3 What do program committees look for?

The program committee looks for interesting, novel, exciting results that significantly enhance our ability to develop and maintain software, to know the quality of the software we develop, to recognize general principles about software, or to analyze properties of software.

You should explain your result in such a way that someone else could use your ideas. Be sure to explain what's novel or original – is it the idea, the application of the idea, the implementation, the analysis, or what?

Define critical terms precisely. Use them consistently. The more formal or analytic the paper, the more important this is.

Here are some questions that the program committee may ask about your paper:

What, precisely, do you claim to contribute?

Does your result fully satisfy your claims? Are the definitions precise, and are terms used consistently?

Authors tend to have trouble in some specific situations. Here are some examples, with advice for staying out of trouble:

- *If your result ought to work on large systems*, explain why you believe it scales.
- *If you claim your method is "automatic"*, using it should not require human intervention. If it's automatic when it's operating but requires manual assistance to configure, say so. If it's automatic except for certain cases, say so, and say how often the exceptions occur.
- *If you claim your result is "distributed"*, it probably should not have a single central controller or server. If it does, explain what part of it is distributed and what part is not.
- *If you're proposing a new notation for an old problem*, explain why your notation is clearly superior to the old one.
- *If your paper is an "experience report"*, relating the use of a previously-reported tool or technique in a practical software project, be sure that you explain what idea the reader can take away from the paper to use in other settings. If that idea is increased confidence in the tool or technique, show how your experience should increase the reader's confidence for applications beyond the example of the paper.

What's new here?

The program committee wants to know what is novel or exciting, and why. What, specifically, is the contribution? What is the increment over earlier work by the same authors? by other authors? Is this a sufficient increment, given the usual standards of subdiscipline?

Above all, the program committee also wants to know what you actually contributed to our store of knowledge about software engineering. Sure, you wrote this tool and tried it out. But was your contribution the technique that is embedded in the tool, or was it making a tool that's more effective than other tools that implement the technique, or was it showing that the tool you described in a previous paper actually worked on a practical large-scale problem? It's better for you as the author to explain than for the program committee to guess. Be clear about your claim ...

Awful	▼	• I completely and generally solved ... (unless you actually did!)
Bad	▼	• I worked on galumphing. (or studied, investigated, sought, explored)
Poor	▼	• I worked on improving galumphing. (or contributed to, participated in, helped with)
Good	▲	• I showed the feasibility of composing blitzing with flitzing. • I significantly improved the accuracy of the standard detector. (or proved, demonstrated, created, established, found, developed)
Better	▲	• I automated the production of flitz tables from specifications. • With a novel application of the blivet transform, I achieved a 10% increase in speed and a 15% improvement in coverage over the standard method.

Use verbs that show results and achievement, not just effort and activity.

> **"*Try* not. *Do*,** or ***do*** not. There is no ***try***." -- Yoda

What has been done before? How is your work different or better?

What existing technology does your research build on? What existing technology or prior research does your research provide a superior alternative to? What's new here compared to your own previous work? What alternatives have other researchers pursued, and how is your work different or better?

As in other areas of science and engineering, software engineering knowledge grows incrementally. Program committees are very interested in your interpretation of prior work in the area. They want to know how your work is related to the prior work, either by building on it or by providing an alternative. If you don't explain this, it's hard for the program committee to understand how you've added to our store of knowledge. You may also damage your credibility if the program committee can't tell whether you know about related work.

Explain the relation to other work clearly ...

Awful	▼	The galumphing problem has attracted much attention [3,8,10,18,26,32,37]
Bad	▼	Smith [36] and Jones [27] worked on galumphing.
Poor	▼	Smith [36] addressed galumphing by blitzing, whereas Jones [27] took a flitzing approach.
Good	▲	Smith's blitzing approach to galumphing [36] achieved 60% coverage [39]. Jones [27] achieved 80% by flitzing, but only for pointer-free cases [16].
Better	▲	Smith's blitzing approach to galumphing [36] achieved 60% coverage [39]. Jones [27] achieved 80% by flitzing, but only for pointer-free cases [16]. We modified the blitzing approach to use the kernel representation of flitzing and achieved 90% coverage while relaxing the restriction so that only cyclic data structures are prohibited.

What, precisely, is the result?

Explain what your result is and how it works. Be concrete and specific. Use examples.

If you introduce a new model, be clear about its power. How general is it? Is it based on empirical data, on a formal semantics, on mathematical principles? How formal is it—a qualitative model that provides design guidance may be as valuable as a mathematical model of some aspect of correctness, but they will have to satisfy different standards of proof. Will the model scale up to problems of size appropriate to its domain?

If you introduce a new metric, define it precisely. Does it measure what it purports to measure and do so better than the alternatives? Why?

If you introduce a new architectural style, design pattern, or similar design element, treat it as if it were a new generalization or model. How does it differ from the alternatives? In what way is it better? What real problem does it solve? Does it scale?

If your contribution is principally the synthesis or integration of other results or components, be clear about why the synthesis is itself a contribution. What is novel, exciting, or nonobvious about the integration? Did you generalize prior results? Did you find a better representation? Did your research improve the individual results or components as well as integrating them? A paper that simply reports on using numerous elements together is not enough, even if it's well-engineered. There must be an idea or lesson or model that the reader can take from the paper and apply to some other situation.

If your paper is chiefly a report on experience applying research results to a practical problem, say what the reader can learn from the experience. Are your conclusions strong and well-supported? Do you show comparative data and/or statistics? An anecdotal report on a single project is usually not enough. Also, if your report mixes additional innovation with validation through experience, avoid confusing your discussion of the innovation with your report on experience. After all, if you changed the result before you applied it, you're evaluating the changed result. And if you changed the result while you were applying it, you may have confounded the experiences with the two versions.

If a tool plays a featured role in your paper, what is the role of the tool? Does it simply support the main contribution, or is the tool itself a principal contribution, or is some aspect of the tool's use or implementation the main point? Can a reader apply the idea without the tool? If the tool is a central part of result, what is the technical innovation embedded in the tool or its implementation?

If a system implementation plays a featured role in your paper, what is the role of the implementation? Is the system sound? Does it do what you claim it does? What ideas does the system demonstrate?

- *If the implementation illustrates an architecture or design strategy*, what does it reveal about the architecture? What was the design rationale? What were the design tradeoffs? What can the reader apply to a different implementation?
- *If the implementation demonstrates an implementation technique*, how does it help the reader use the technique in another setting?
- *If the implementation demonstrates a capability or performance improvement*, what concrete evidence does it offer to support the claim?
- *If the system is itself the result*, in what way is it a contribution to knowledge? Does it, for example, show you can do something that no one has done before (especially if people doubted that this could be done)?

4. Why should the reader believe your result?

Show evidence that your result is valid—that it actually helps to solve the problem you set out to solve.

4.1. What kinds of validation do software engineers do?

Software engineers offer several kinds of evidence in support of their research results. It is essential to select a form of validation that is appropriate for the type of research result and the method used to obtain the result. As an obvious example, a formal model should be supported by rigorous derivation and proof, not by one or two simple examples. On the other hand, a simple example derived from a practical system may play a major role in validating a new type of development method. Table 5 lists the types of research validation that are used in software engineering research papers and provides specific examples. In this table, the examples are keyed to the type of result they apply to.

Table 5. Types of software engineering research validation

Type of validation	Examples
Analysis	I have analyzed my result and find it satisfactory through rigorous analysis, e.g. ... For a formal model ... rigorous derivation and proof For an empirical model ... data on use in controlled situation For a controlled experiment ... carefully designed experiment with statistically significant results
Evaluation	Given the stated criteria, my result... For a descriptive model ... adequately describes phenomena of interest ... For a qualitative model ... accounts for the phenomena of interest... For an empirical model ... is able to predict ... because ..., or ... generates results that fit actual data ... Includes feasibility studies, pilot projects
Experience	My result has been used on real examples by someone other than me, and the evidence of its correctness/usefulness/effectiveness is ... For a qualitative model ... narrative For an empirical model or tool ... data, usually statistical, on practice For a notation or technique ... comparison of systems in actual use
Example	Here's an example of how it works on For a technique or procedure ...a "slice of life" example based on a real system ... For a technique or procedure ...a system that I have been developing ... For a technique or procedure ... a toy example, perhaps motivated by reality The "slice of life" example is most likely to be convincing, especially if accompanied by an explanation of why the simplified example retains the essence of the problem being solved. Toy or textbook examples often fail to provide persuasive validation, (except for standard examples used as model problems by the field).
Persuasion	I thought hard about this, and I believe passionately that ... For a technique ... if you do it the following way, then ... For a system ... a system constructed like this would ... For a model ... this example shows how my idea works Validation purely by persuasion is rarely sufficient for a research paper. Note, though, that if the original question was about feasibility, a working system, even without analysis, can suffice
Blatant assertion	No serious attempt to evaluate result. This is highly unlikely to be acceptable

4.2 Which of these are most common?

Alas, well over a quarter of the ICSE 2002 abstracts give no indication of how the paper's results are validated, if at all. Even when the abstract mentions that the result was applied to an example, it was not always clear whether the example was a textbook example, or a report on use in the field, or something in between.

The most successful kinds of validation were based on analysis and real-world experience. Well-chosen examples were also successful. Persuasion was not persuasive, and narrative evaluation was only slightly more successful. Table 6 gives the distribution of submissions to ICSE 2002, based on reading the abstracts (but not the papers), followed by graphs of the counts and distributions. Figures 5 and 6 show these counts and distributions.

Table 6. Types of research validation represented in ICSE 2002 submissions and acceptances			
Type of validation	**Submitted**	**Accepted**	**Ratio Acc/Sub**
Analysis	48 (16%)	11 (26%)	23%
Evaluation	21 (7%)	1 (2%)	5%
Experience	34 (11%)	8 (19%)	24%
Example	82 (27%)	16 (37%)	20%
Some example, can't tell whether it's toy or actual use	6 (2%)	1 (2%)	17%
Persuasion	25 (8%)	0 (0.0%)	0%
No mention of validation in abstract	84 (28%)	6 (14%)	7%
TOTAL	300(100.0%)	43 (100.0%)	14%

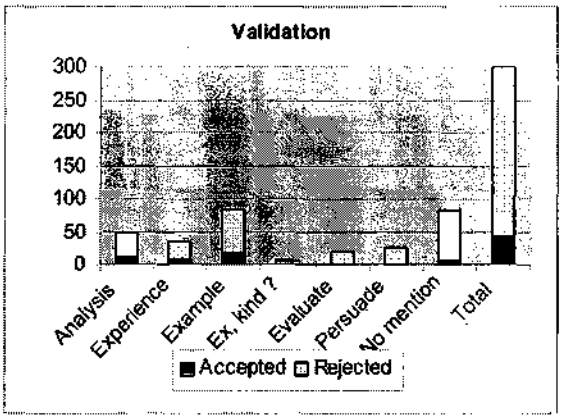


Figure 5. Counts of acceptances and rejections by type of validation

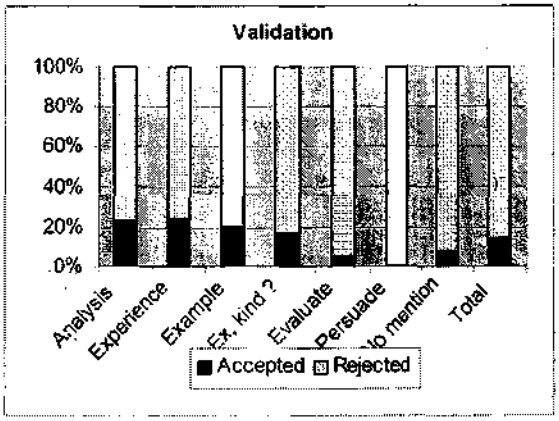


Figure 6. Distribution of acceptances and rejections by type of validation

4.3 What do program committees look for?

The program committee looks for solid evidence to support your result. It's not enough that your idea works for you, there must also be evidence that the idea or the technique will help someone else as well.

The statistics above show that analysis, actual experience in the field, and good use of realistic examples tend to be the most effective ways of showing why your result should be believed. Careful narrative, qualitative analysis can also work if the reasoning is sound.

Why should the reader believe your result?

Is the paper argued persuasively? What evidence is presented to support the claim? What kind of evidence is offered? Does it meet the usual standard of the subdiscipline?

Is the kind of evaluation you're doing described clearly and accurately? "Controlled experiment" requires more than data collection, and "case study" requires more than anecdotal discussion. Pilot studies that lay the groundwork for controlled experiments are often not publishable by themselves.

Is the validation related to the claim? If you're claiming performance improvement, validation should analyze performance, not ease of use or generality. And conversely.

Is this such an interesting, potentially powerful idea that it should get exposure despite a shortage of concrete evidence?

Authors tend to have trouble in some specific situations. Here are some examples, with advice for staying out of trouble:

- *If you claim to improve on prior art,* compare your result objectively to the prior art.
- *If you used an analysis technique,* follow the rules of that analysis technique. If the technique is not a common one in software engineering (e.g., meta-analysis, decision theory, user studies or other behavioral analyses), explain the technique and standards of proof, and be clear about your adherence to the technique.
- *If you offer practical experience as evidence for your result,* establish the effect your research has. If at all possible, compare similar situations with and without your result.

- *If you performed a controlled experiment,* explain the experimental design. What is the hypothesis? What is the treatment? What is being controlled? What data did you collect, and how did you analyze it? Are the results significant? What are the potentially confounding factors, and how are they handled? Do the conclusions follow rigorously from the experimental data?
- *If you performed an empirical study,* explain what you measured, how you analyzed it, and what you concluded. What data did you collect, and how? How is the analysis related to the goal of supporting your claim about the result? Do not confuse correlation with causality.
- *If you use a small example for explaining the result,* provide additional evidence of its practical use and scalability.

5. How do you combine the elements into a research strategy?

It is clear that not all combinations of a research question, a result, and a validation strategy lead to good research. Software engineering has not developed good general guidance on this question.

Tables 1, 3, and 5 define a 3-dimensional space. Some portions of that space are densely populated: One common paradigm is to find a better way to perform some software development or maintenance task, realize this in a concrete procedure supported by a tool, and evaluate the effectiveness of this procedure and tool by determining how its use affects some measure (e.g., error rates) of quality. Another common paradigm is to find a better way to evaluate a formalizable property of a software system, develop a formal model that supports inference, and to show that the new model allows formal analysis or proof of the properties of interest.

Clearly, the researcher does not have free choice to mix and match the techniques—validating the correctness of a formal model through field study is as inappropriate as attempting formal verification of a method based on good organization of rules of thumb.

Selecting a type of result that will answer a given question usually does not seem to present much difficulty, at least for researchers who think carefully about the choice. Blindly adopting the research paradigm someone used last year for a completely different problem is a different case, of course, and it can lead to serious misfits.

Choosing a good form of validation is much harder, and this is often a source of difficulty in completing a successful paper. Table 6 shows some common good matches. This does not, unfortunately, provide complete guidance.

When I advise PhD students on the validation section of their theses, I offer the following heuristic: Look carefully at the short statement of the result—the principal claim of the thesis. This often has two or three clauses (e.g., I found an efficient and complete method ..."); if so, each presents a separate validation problem. Ask of each clause whether it is a global statement ("always", "fully"), a qualified statement ("a 25% improvement", "for noncyclic structures..."), or an existential statement {"we found an instance of"). Global statements often require analytic validation, qualified statements can often be validated by evaluation or careful examination of experience, and existential statements can sometimes be validated by a single positive example. A frequent result of this discussion is that students restate the thesis claims to reflect more precisely what the theses actually achieve. If we have this discussion early enough in the thesis process, students think about planning the research with demonstrable claims in mind.

Concretely, Table 7 shows the combinations that were represented among the accepted papers at ICSE 2002, omitting the 7 for which the abstracts were unclear about validation:

Table 7. Paradigms of ICSE2002 acceptances

Question	Result	Validation	#
Devel method	Procedure	Analysis	2
Devel method	Procedure	Experience	3
Devel method	Procedure	Example	3
Devel method	Qual model	Experience	2
Devel method	Analytic model	Experience	2
Devel method	Notation or tool	Experience	1
Analysis method	Procedure	Analysis	5
Analysis method	Procedure	Evaluation	1
Analysis method	Procedure	Experience	2
Analysis method	Procedure	Example	6
Analysis method	Analytic model	Experience	1
Analysis method	Analytic model	Example	2
Analysis method	Tool	Analysis	1
Eval of instance	Specific analysis	Analysis	3
Eval of instance	Specific analysis	Example	2

6. Does the abstract matter?

The abstracts of papers submitted to ICSE convey a sense of the kinds of research submitted to the conference. Some abstracts were easier to read and (apparently) more informative than others. Many of the clearest abstracts had a common structure:

- Two or three sentences about the current state of the art, identifying a particular problem
- One or two sentences about what this paper contributes to improving the situation

- One or two sentences about the specific result of the paper and the main idea behind it
- A sentence about how the result is demonstrated or defended

Abstracts in roughly this format often explained clearly what readers could expect in the paper.

Acceptance rates were highest for papers whose abstracts indicate that analysis or experience provides evidence in support of the work. Decisions on papers were made on the basis of the whole papers, of course, not just the abstracts—but it is reasonable to assume that the abstracts reflect what's in the papers.

Whether you like it or not, people judge papers by their abstracts and read the abstract in order to decide whether to read the whole paper. It's important for the abstract to tell the story. Don't assume, though, that simply adding a sentence about analysis or experience to your abstract is sufficient; the paper must deliver what the abstract promises

7. Questions you might ask about this report

7.1. Is this a sure-fire recipe?

No, not at all. First, it's not a recipe. Second, not all software engineers share the same views of interesting and significant research. Even if your paper is clear about what you've done and what you can conclude, members of a program committee may not agree about how to interpret your result. These are usually honest technical disagreements, and committee members will try hard to understand what you have done. You can help by explaining your work clearly; this report should help you do that.

7.2 Is ICSE different from other conferences?

ICSE recognizes several distinct types of technical papers [6]. For 2002, they were published separately in the proceedings

Several other conferences offer "how to write a paper" advice:

In 1993, several OOPSLA program committee veterans gave a panel on "How to Get a Paper Accepted at OOPSLA" [9]. This updated the 1991 advice for the same conference [14]

SIGSOFT offers two essays on getting papers accepted, though neither was actually written for a software engineering audience. They are "How to Have Your Abstract Rejected" [26] (which focuses on theoretical papers) and "Advice to Authors of Extended Abstracts", which was written for PLDI. [16].

Rather older, Levin and Reddell, the 1983 SOSP (operating systems) program co-chairs offered advice on writing a good systems paper [11]. USENIX now provides this advice to its authors. Also in the systems vein, Partridge offers advice on "How to Increase the Chances Your Paper is Accepted at ACM SIGCOMM" [15].

SIGCHI offers a "Guide to Successful Papers Submission" that includes criteria for evaluation and discussion of common types of CHI results, together with how different evaluation criteria apply for different types of results [13]. A study [8] of regional factors that affect acceptance found regional differences in problems with novelty, significance, focus, and writing quality.

In 1993, the SIGGRAPH conference program chair wrote a discussion of the selection process, "How to Get Your SIGGRAPH Paper Rejected" [10]. The 2003 SIGGRAPH call for papers [21] has a description of the review process and a frequently-asked questions section with an extensive set of questions on "Getting a Paper Accepted".

7.3. What about this report itself?

People have asked me, "what would happen if you submitted this to ICSE?" Without venturing to predict what any given ICSE program committee would do, I note that as a research result or technical paper (a "finding" in Brooks' sense [3]) it falls short in a number of ways:

- There is no attempt to show that anyone else can apply the model. That is, there is no demonstration of inter-rater reliability, or for that matter even repeatability by the same rater.
- The model is not justified by any principled analysis, though fragments, such as the types of models that can serve as results, are principled. In defense of the model, Bowker and Starr [2] show that useful classifications blend principle and pragmatic descriptive power.
- Only one conference and one program committee is reflected here.
- The use of abstracts as proxies for full papers is suspect.
- There is little discussion of related work other than the essays about writing papers for other conferences. Although discussion of related work does appear in two complementary papers [19, 20], this report does not stand alone.

On the other hand, I believe that this report does meet Brooks' standard for "rules of thumb" (generalizations, signed by the author but perhaps incompletely supported by data, judged by usefulness and freshness), and I offer it in that sense.

8. Acknowledgements

This work depended critically on access to the entire body of submitted papers for the ICSE 2002 conference,

which would not have been possible without the cooperation and encouragement of the ICSE 2002 program committee. The development of these ideas has also benefited from discussion with the ICSE 2002 program committee, with colleagues at Carnegie Mellon, and at open discussion sessions at FSE Conferences. The work has been supported by the A. J. Perlis Chair at Carnegie Mellon University.

9. References

1. Victor R. Basili. The experimental paradigm in software engineering. In *Experimental Software Engineering Issues: Critical Assessment and Future Directives*. Proc of Dagstuhl-Workshop, H. Dieter Rombach, Victor R. Basili, and Richard Selby (eds), published as *Lecture Notes in Computer Science #706*, Springer-Verlag 1993.
2. Geoffrey Bowker and Susan Leigh Star: *Sorting Things Out: Classification and Its Consequences*. MIT Press, 1999
3. Frederick P. Brooks, Jr. Grasping Reality Through Illusion—Interactive Graphics Serving Science. *Proc 1988 ACM SIGCHI Human Factors in Computer Systems Conf (CHI '88)* pp. 1-11.
4. Rebecca Burnett. *Technical Communication*. Thomson Heinle 2001.
5. Thomas F. Gieryn. *Cultural Boundaries of Science: Credibility on the line*. Univ of Chicago Press, 1999.
6. ICSE 2002 Program Committee. *Types of ICSE papers*. http://icse-conferences.org/2002/info/paperTypes.html
7. Impact Project. "Determining the impact of software engineering research upon practice. Panel summary, *Proc. 23rd International Conference on Software Engineering (ICSE 2001)*, 2001
8. Ellen Isaacs and John Tang. *Why don't more non-North-American papers get accepted to CHI?* http://acm.org/sigchi/bulletin/1996.1/isaacs.html
9. Ralph E. Johnson & panel. How to Get a Paper Accepted at OOPSLA. *Proc OOPSLA'93*, pp. 429-436, http://acm.org/sigplan/oopsla/oopsla96/how93.html
10. Jim Kajiya. How to Get Your SIGGRAPH Paper Rejected. Mirrored at http://www.cc.gatech.edu/student.services/phd/phd-advice/kajiya
11. Roy Levin and David D. Redell. How (and How Not) to Write a Good Systems Paper. *ACM SIGOPS Operating Systems Review*, Vol. 17, No. 3 (July, 1983), pages 35-40. http://ftp.digital.com/pub/DEC/SRC/other/SOSPadvice.txt
12. William Newman. A preliminary analysis of the products of HCI research, using pro forma abstracts. *Proc 1994 ACM SIGCHI Human Factors in Computer Systems Conf (CHI '94)*, pp.278-284.
13. William Newman et al. *Guide to Successful Papers Submission at CHI 2001*. http://acm.org/sigs/sigchi/chi2001/call/submissions/guide-papers.html
14. OOPSLA '91 Program Committee. How to get your paper accepted at OOPSLA. *Proc OOPSLA'91*, pp.359-363. http://acm.org/sigplan/oopsla/oopsla96/how91.html
15. Craig Partridge. How to Increase the Chances your Paper is Accepted at ACM SIGCOMM. http://www.acm.org/sigcomm/conference-misc/author-guide.html
16. William Pugh and PDLI 1991 Program Committee. *Advice to Authors of Extended Abstracts*. http://acm.org/sigsoft/conferences/pughadvice.html
17. Samuel Redwine, et al. *DoD Related Software Technology Requirements, Practices, and Prospects for the Future*. IDA Paper P-1788, June 1984.
18. S. Redwine & W. Riddle. Software technology maturation. *Proceedings of the Eighth International Conference on Software Engineering*, May 1985, pp. 189-200.
19. Mary Shaw. The coming-of-age of software architecture research. *Proc. 23rd Int'l Conf on Software Engineering (ICSE 2001)*, pp. 656-664a.
20. Mary Shaw. What makes good research in software engineering? Presented at ETAPS 02, appeared in Opinion Corner department, *Int'l Jour on Software Tools for Tech Transfer*, vol 4, DOI 10.1007/s10009-002-0083-4, June 2002.
21. SigGraph 2003 Call for Papers. http://www.siggraph.org/s2003/cfp/papers/index.html
22. W. F. Tichy, P. Lukowicz, L. Prechelt, & E. A. Heinz. "Experimental evaluation in computer science: A quantitative study." *Journal of Systems Software*, Vol. 28, No. 1, 1995, pp. 9-18.
23. Walter F. Tichy. "Should computer scientists experiment more? 16 reasons to avoid experimentation." *IEEE Computer*, Vol. 31, No. 5, May 1998
24. Marvin V. Zelkowitz and Delores Wallace. Experimental validation in software engineering. *Information and Software Technology*, Vol 39, no 11, 1997, pp. 735-744.
25. Marvin V. Zelkowitz and Delores Wallace. Experimental models for validating technology. *IEEE Computer*, Vol. 31, No. 5, 1998, pp.23-31.
26. Mary-Claire van Leunen and Richard Lipton. *How to have your abstract rejected.* http://acm.org/sigsoft/conferences/vanLeunenLipton.html

Subject faculty 17/2/2004

HOD