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

IV/II

Course Name with Code : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **I** **Date of Lecture:**

Topic of Lecture: INTRODUCTION: SOFTWARE LIFE-CYCLE ACTIVITIES

Introduction : (Maximum 5 sentences) :

- Software engineering aims to significantly improve software productivity and software quality while reducing software costs and time to market.
- Software engineering consists of three tracks of interacting life cycle activities software development, software quality assurance; and software project management activities.
- Object-oriented (OO) software engineering is a specialization of software engineering.
- It views the world and systems as consisting of objects that interact with each other.

Prerequisite knowledge for Complete understanding and learning of Topic:

Basic of Software Engineering

Detailed content of the Lecture:

WHAT IS SOFTWARE ENGINEERING?

- Software systems are complex intellectual products. Software development must ensure that the software system satisfies its requirements, the budget is not overrun, and the system is delivered according to schedule.
- Software engineering as a discipline is focused on the research, education, and application of engineering processes and methods to significantly increase software productivity and software quality while reducing software costs and time to market.
- The overall objectives of software engineering are significantly increasing software productivity (P) and qualify (QJ while reducing software production and operating costs (C) and time to market (T). These objectives are abbreviated as PQCT.
- Research, education, and application of software engineering processes and-methods are the means to accomplish these goals.
- These processes and methods are classified into three sets of activities: development, quality assurance, and project management activities.
- The development activities transform an initial system concept into an operational system.
- The quality assurance activities ensure that the development activities are carried out correctly and that the artifacts produced by the activities are correct.

- These ensure that the desired software system is produced and delivered.
- Project management activities plan for the project, schedule and allocate resources to the development and quality assurance activities, and ensure that the system is developed and delivered on time and within budget.

WHY IS SOFTWARE ENGINEERING?

- First, software is expanding into all sectors of our society. Companies rely on software to run and expand their businesses.
- Software systems are getting larger- and more complex. Today, it is common to develop systems that contain millions of lines of source code.
- For many embedded systems, software cost has increased to 90%-95% of the total system cost from 5%-10% two decades ago.
- Some embedded systems use application specific integrated circuits (ASIC) and firmware. These are integrated circuits with the software burned into the hardware.
- They are costly to replace; and hence, the quality of the software is critical. These call for a software engineering approach to system development.
- Second, software engineering supports teamwork, which is needed for large system development. Large software systems require considerable effort to design, implement, and test.
- A typical software engineer can produce average 50-100 lines of source code per day.
- This includes the time required to perform analysis, design, implementation, integration, and testing.
- Thus, a small system of 10000 lines of code would require one software engineer to work between 100 and 200 days or 5 to 10 months.
- A medium-size system of 500,000 lines of source cede, would require a software engineer to work 5,000 to 10,000days or 20 to 4(1 years. It is not acceptable for most businesses to wait this long for their systems.
- Therefore, real world software systems must be designed and implemented by a team or teams of software engineers.
- Therefore, real world software systems must be designed and implemented by a team or teams of software engineers. For example, a medium-size software system requires 20 to 40 software engineers to work for one year.
- When two or more software engineers work together to develop a software system, serious conceptualization, communication, and coordination problems arise.
- Conceptualization is the process of observing and classifying real-world phenomena to form a mental model to help understand the application for which the system is built.
- Conceptualization is a challenge for teamwork because the software engineers may perceive the world differently due to differences in their education, cultural backgrounds, career experiences, assumptions, and other factors.
- The ancient story about four blind men and an elephant illustrates this problem. The four blind men wanted to know what an elephant looked like.
- They obtained permission to touch the elephant. One blind man touched one leg of the elephant and said that an elephant was like a tree trunk.
- The other three touched the elephant's stomach, tail, and car respectively. They said that an elephant was like a Wall, a rope, and a fan.
- We as software developers arc like the four blind men trying to perceive or understand an application.
- If the developers perceive the application differently, then how can they design and implement software components to work with each other?

- Software engineering provides a solution, when a team of software engineers work together, they need to exchange their understanding and design ideas. However, the natural language is too informal and often leads 10 misunderstanding.
- Software engineering provides the Unified Modeling Language (UML) for software engineers to communicate their ideas. Finally, when teams of software engineers work together, how can they collaborate and coordinate their efforts?
- For example how do they divide the work ad assign the pieces to the teams and team members? How do they integrate the components designed and implemented by different teams and team members? Again, software engineering provides a solution.

Video Content / Details of website for further learning (if any):
https://www.guru99.com/what-is-software-engineering.html
https://www.youtube.com/watch?v=WxkP5KR_Emk

Important Books/Journals for further learning including the page nos.:
McGraw-Hill Education,'' Object-Oriented Software Engineering: An Agile Unified Methodology", 2013,T1 (4-10)

Course Faculty

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

L-2

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **I** **Date of Lecture:**

Topic of Lecture: SOFTWARE LIFE-CYCLE ACTIVITIES

Introduction : (Maximum 5 sentences) :

- First, software is expanding into all sectors of our society. Companies rely on software to run and expand their businesses. Software systems are getting larger- and more complex.
- Second, software engineering supports teamwork, which is needed for large system development. Large software systems require considerable effort to design, implement, and test.
- A typical software engineer can produce average 50-100 lines of source code per day. This includes the time required to perform analysis, design, implementation. integration, and testing.

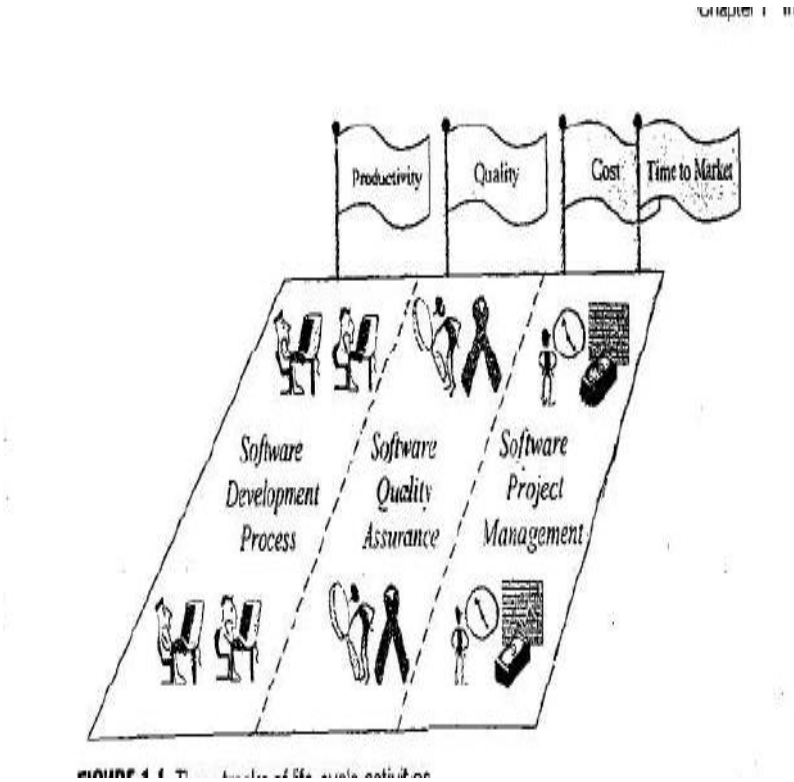
Prerequisite knowledge for Complete understanding and learning of Topic:

- Software Engineering
- Software Development

Detailed content of the Lecture:

- Software engineering focuses on three tracks of activities as exhibits. These activities take place simultaneously throughout the software life cycle.
- Software development process. A software development process transforms the initial system concept into the operational system running in the target environment.
- It identifies the business needs, conducts a feasibility study, and formulates the requirements or capabilities that the system must deliver.
- It also designs, implements, tests, and deploys the system to the target environment.
- Software quality assurance. Software quality assurance (SQA) ensures that the development activities are performed properly, and the software artifacts produced by the development activities meet the software requirements and desired quality standards.

- Software project management. Software project management oversees the control and administration of the development and SQA activities.
- Project management activities include effort estimation. Project planning and scheduling, risk management, and project administration, among others.
- These activities ensure that the software system is delivered on time and within budget.

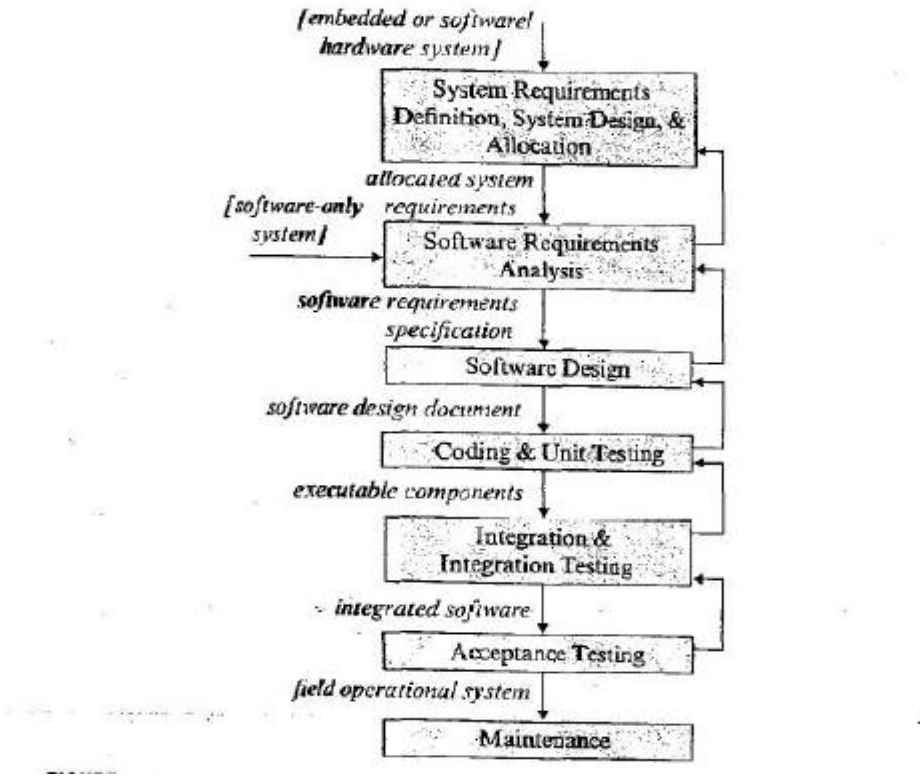


Software Development Process

- A software development process is often called a software process. The need for a process is similar to custom home construction and many others major undertakings.
- The activities of custom home construction include acquisition of home buyer requirements, custom home design, build, inspection, and delivery.
- A software process consists of a series of phases of activities performed to produce the software system.
- In some cases, the software system is a part of a larger system.

System Requirements Definition, System Design, and Allocation

- These are system engineering activities often performed for embedded systems.
- System requirements definition identifies the capabilities for the total system and formulates them as system requirements.



Software Requirements Analysis

- Software requirements analysis refines the system requirements allocated to the software system. It also identifies other capabilities for the software system. These and the refined system requirements arc specified in a software requirements specification (SRS).

Software design

- Software design determines the software architecture, or the overall structure, of the software system.
- It specifies the subsystems, their relationships the subsystems functions, interfaces, and how the subsystems interact with each other.
- Design of the user interface is another important activity of software design.
- That is, it depicts the look and feel of the windows and dialogs, and describes how the system interacts with the users.

Software Implementation, Testing, and Maintenance.

- During the implementation and unit testing phase, programs are written to implement the design. The programs are tested, and reviewed by peers to ensure correctness and compliance to coding standards.
- During the integration phase, the program modules are integrated and tested to ensure that they work with each other.
- During acceptance testing, test cases are designed and run to check that the software indeed

satisfies the software requirements.

Software project Management

Software project management activities ensure that the software system under development will be delivered on schedule and within the budget constraint.

Effort estimation.

Effort estimation derives the human resources and durations required to perform the development and SQA activities.

Project planning and scheduling.

- Project planning and scheduling are aimed at producing an overall plan for the project, The project plan will guide the project teams throughout the life-cycle process.

- Risk management. Many events could jeopardize a project. For example, a management person or a key technical staff leaves the project, or the project is far behind schedule. These are called risk items.
- Project administration. Project administration is an ongoing function of project management. It performs the management activities as specified in the project plan.
- Software configuration management. During the development process, numerous software artifacts are produced. These include requirements specification, software design, code, test cases, user's manual, and the like. These compose the software, or part of it, under different stages of the development process.

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

https://www.tutorialspoint.com/sdlc/sdlc_overview.htm

https://www.cleverism.com/software-development-life-cycle-sdlc-methodologies/

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

McGraw-Hill Education,'' Object-Oriented Software Engineering: An Agile Unified Methodology", 2013, Page No (4-10)

Course Faculty

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 : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **I** **Date of Lecture:**

Topic of Lecture: OBJECT ORIENTED SOFTWARE ENGINEERING

Introduction : (Maximum 5 sentences) :

- First, software is expanding into all sectors of our society. Companies rely on software to run and expand their businesses. Software systems are getting larger- and more complex.
- Today, it is common to develop systems that contain millions of lines of source code. For many embedded systems, software cost has increased t0l90%-95% of the total system cost from 5%-10% two decades ago.
- Second, software engineering supports teamwork, which is needed for large system development.
- Large software systems require considerable effort to design, implement, and test. A typical software engineer can produce average 50-100 lines of source code per day. This includes the time required to perform analysis, design, implementation. integration, and testing.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Software Engineering
- Software Life Cycle

OO Modeling and design languages

- 00 *modeling and design languages* for the team members to communicate their analysis and design ideas: A modeling and design language defines the notions and notations as well as rules for using the notations

- The Unified Modeling Language (UML) [36] is the most widely used00 modeling and design language.

00 software development processes

- 00 software development processes to guide the development effort. The unified process (UP)

is a well-known development process while agile processes have emerged in recent years.

- 00 software development methodologies that detail the steps or how to carry out the activities of a software process.

00 development tools and environments

- There are commercial products as well as public domain software. For example, the Net Beans integrated development environment (IDE) is a free, open source software. It comes with a bundle of plugins that support activities of the entire software development life cycle.

Object-Oriented Modeling and Design Languages

- The rapid spread of C++ in the 1980s motivated the need for a development methodology to guide 00 software development efforts. Three influential 00 development methodologies, among many others were proposed and widely used in the software industry.

- These are Booch Diagram, Object Modeling Technique (OMT) and Use Case Engineering.
- The industry soon discovered that it was a monumental challenge to integrate systems designed and implemented using different methodologies.
- The reason is that different methodologies use different modeling concepts and notations. To solve this problem, the Object Management Group (OMG) adopted the Unified Modeling Language (UML) as an OMG standard.
- UML diagrams are used in the requirements analysis phase to help the development team understand the business of the existing application.
- They are used in the design phase as part of the design specification. UML diagrams will be presented throughout the rest of this book.

Object-Oriented Development Process

- The sequential nature of the waterfall process implies that changes to the requirements are difficult and costly**,** This is because any change to the requirements affects the design and implementation; these must be changed as well.
- The long development duration of the waterfall process implies that the system is dated as soon as it is released.
- To overcome these problems, several software process models have been proposed. All of these adopt an iterative, rather than a strictly sequential, process of development actives. Examples are the spiral process, the unified process, and agile processes.

Object-Oriented Development Methodologies

- A software process speci fies "when to do what," but not "how to do them" That-is it defines the development activities but not how to perform the activities.
- UML is a modeling language. It lets the software engineers describe their analysis and design ideas using the diagrams. It does not help the software engineers to produce the analysis and design ideas.
- A software development methodology fills the gap. It specifies the steps and how to perform the steps to carry out the activities of a software process. Conventional 00 development methodologies include Booch Diagram, Object Modeling Techniques (OMT). use Case Engineering. and other methods.
- Agile methods include Serum, Dynamic System Development Method (DSDM). Feature Driven Development (FDD), Crystal Clear, Extreme Programming (XP), Lean Development Method; and others.

00 Replace the Conventional Approaches

- First, maintaining numerous conventional systems is required. Second, numerous organizations still use the conventional approaches.
- Third, a conventional methodology may be more appropriate for some projects such as scientific computing. Finally, a system may consist of components developed by conventional and 00 approaches. Therefore, 00 and conventional approaches will coexist for many years

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

http://cs-exhibitions.uni-klu.ac.at/index.php?id=448

https://www.youtube.com/watch?v=BqVqjJq7_vI

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

McGraw-Hill Education,'' Object-Oriented Software Engineering: An Agile Unified Methodology", 2013, Page No (11-13)

Course Faculty

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 : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **I** **Date of Lecture:**

Topic of Lecture: SOFTWARE PROCESS

Introduction : (Maximum 5 sentences) :

- *Software process* is a series of phases of activities performed to' construct a software system.
- Each phase produces some artifacts which are the input to other phases. Each phase has a set of entrance criteria and a set of exit criteria.
- Software development is not a scientific process-in other words, many decisions are not made scientifically but politically and economically.
- For example, a good enough algorithm is chosen instead of an optimal one because it is more economical to implement, use, and maintain the good-enough algorithms.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Software Engineering
- OOSE

Detailed content of the Lecture:

SOFTWARE PROCESS MODELS

- Problems with the waterfall process have led researchers and funding agencies to find a better process that considers the wicked properties of software development.

Prototyping Process

- The prototyping process model recognizes the mismatch between the- newly constructed software system and users' expectations, and the challenge to deliver the capabilities within the time and budget constraints.
- As a solution, a prototype of the system is constructed and-used to acquire and validate requirements. Prototypes are also used in feasibility studies as well as design validation.

Evolutionary Process

- Prototypes help requirements acquisition, requirements validation, feasibility study, and validation of design ideas.
- However, throwaway prototypes imply that much effort is wasted. This is true when sophisticated prototypes are needed for feasibility study and design validation of large, real-time embedded systems.
- The evolutionary process model is aimed at solving this problem by letting the prototype evolves. It lets the users experiment with an initial prototype, constructed according to a set of preliminary requirements.

Personal Software Process

- The personal software process (PSP) is a comprehensive framework that is designed to train individual software engineers to improve their personal software processes.
- PSP consists of a series of scripts, forms, standards, and guidelines that the software engineer can apply to carry out a number of predefined programming exercises.

The PSP Process/Evolution

- To facilitate learning, the PSP uses an evolutionary approach. That is, the framework is presented in a series of predefined processes, named PSPO, PSPO.I, PSP1, PSPl.l, PSP2, PSP2.1 and PSP3.0.
- Each of these processes introduces a couple of good software engineering techniques or practices.

PSPO and PSPO.l.

- These two processes introduce process discipline and measurement.
- In particular, PSPO introduces the baseline process, time recording, defect recording, and defect type standard. PSPO.I introduces ceding standard, size measurement, and process improvement proposal.

PSP1 and PSP1.1.

- These two processes introduce estimations and planning.
- In particular, PSPI introduces size estimation and test report while PSPI.I covers planning and scheduling.

PSP2 and PSP2.1.

- These two processes introduce quality management and design.
- In particular, PSP2 presents code review and design review; and PSP2.1 introduces and design template.

PSP3.0.

- This process is designed to guide the development of component-level programs.

PSP Script

- In PSP, all processes are described using *process scripts* or scripts for short.
- Each script specifies the purpose, the entry criteria, the steps or activities of the process, and the exit criteria.

Spiral Process

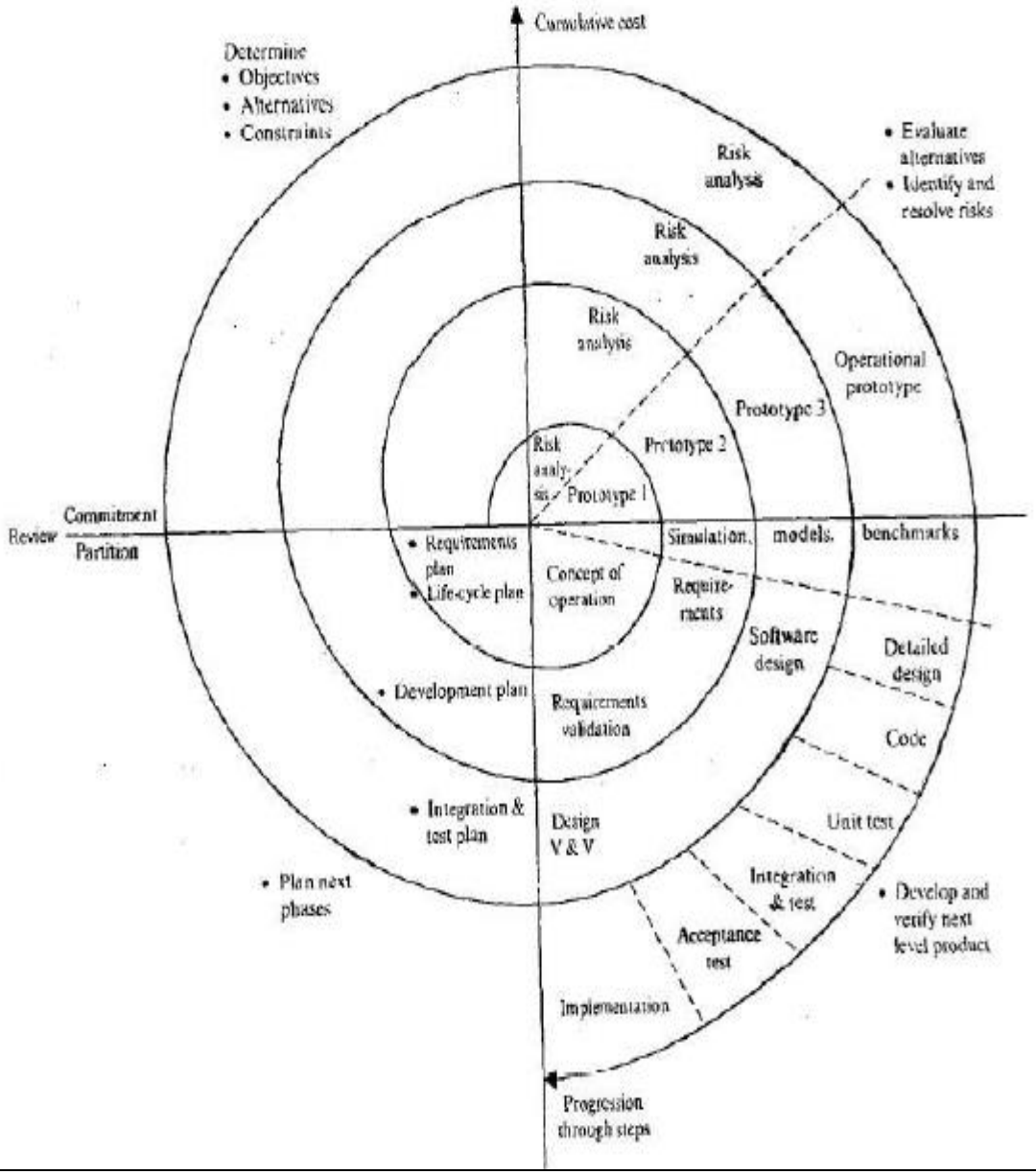
- The spiral process proposed by Barry, Boehm is known for its unique feature for risk management.
- Each cycle of the spiral is aimed at enhancing a certain aspect of the system under development.
- For example, functionality, performance or quality.

1. Determine the objectives alternatives, and constraints/or the current cycle (the Northwest corner of the spiral).

2. Evaluate alternatives, identify and evolve risks (the northeast corner of the spiral).

3.Develop and verify next level system

4. Plan next phases



Agile Processes

- The waterfall process works well for tame problems because such problems possess a number of nice properties.
- Application software development is a wicked problem. It needs a process that is designed to solve wicked problems.

Agile Manifesto

According to the Agile Manifesto,' agile development values four aspects of software development practices, which are different from their conventional plan-driven counterparts. These are listed and explained below.

- Agile development values individuals and interactions over processes and tools.
- Agile development values working software over comprehensive documentation.
- Agile development values customer collaboration over contract negotiation.
- Agile development values responding w change over fallowing a plan.

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

https://www.geeksforgeeks.org/software-processes-in-software- engineering/
https://www.youtube.com/watch?v=YMbAdgb6pG8

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

McGraw-Hill Education,'' Object-Oriented Software Engineering: An Agile Unified Methodology", 2013, Page No (17-18)

Course Faculty

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 : Object Oriented Software Engineering-16CSD04

Course Faculty :

Unit : I **Date of Lecture:**

Topic of Lecture: SOFTWARE DEVELOPMENT METHODOLOGY

Introduction : (Maximum 5 sentences) :

- Software development requires not only a process but also a methodology or development method. Unfortunately, the term "methodology" is often left undefined. This leads to a certain degree of confusion. For example, methodology is often confused with process.
- Process and methodology are important concepts of software engineering. Second, software engineering supports teamwork, which is needed for large system development.
- Large software systems require considerable effort to design, implement, and test. A typical software engineer can produce average 50-100 lines of source code per day. This includes the time required to perform analysis, design, implementation, integration, and testing.

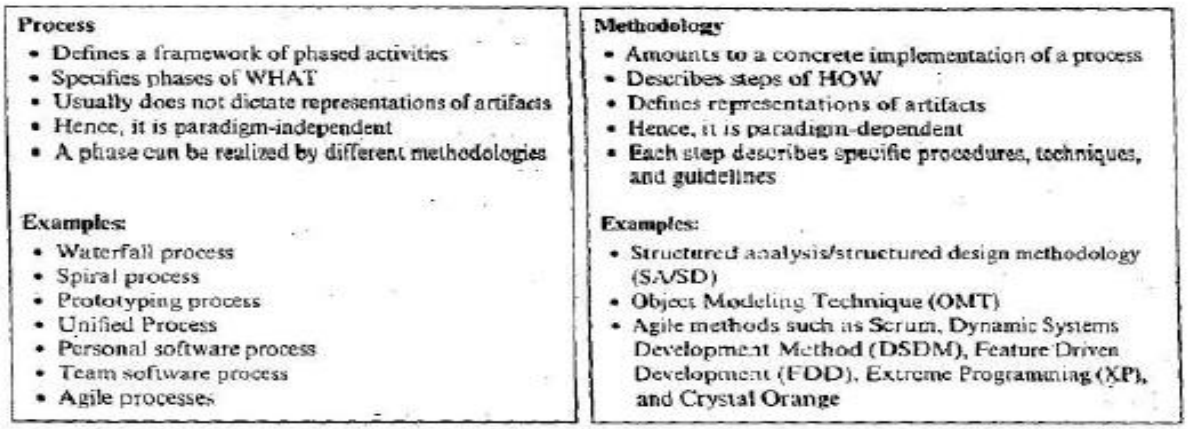
Prerequisite knowledge for Complete understanding and learning of Topic:

- Software Life Cycle
- Process and methodology

Detailed content of the Lecture:

Difference between Process and Methodology

- A software process defines the phased activities or *what to do* in each phase, it does not specify how to perform the activities.
- A software methodology defines the detailed steps or *how to carry out* the activities of a process.
- A software process specifies the input and output of each phase, but it does not dictate the representations of the input and output.
- A methodology defines the steps, step entrance, and exit criteria, and relationships between the steps.



Structured Methodologies

- Structured analysis uses data flow diagrams (DFOs) to model the business processes of real-world applications.
- A DFD is essentially a directed graph, in which the vertexes represent external entities, business processes, and data stores while the directed edges represent data flows between them.
- Divide-and-conquer is employed during structured analysis to decompose complex business processes into lower-level data flow diagrams.
- The steps of structured analysis begin with the construction of a top-level OFD, called the context diagram.
- It depicts the system as the sole process, which interacts with external entities and external data stores. The next steps repeatedly decompose complex processes into simpler processes.
- This is because the relationships between the processes of a DFD are data flow relationships while the relationships between the software modules are control flow relationships.
- The so-called structured design fills the gap.

Classical OO Methodologies

- Before UML, there were classical 00 methodologies, with three of them widely known.
- They are the Booch Method, the Object Modeling Technique (GMT), and the Use case driven approach.
- These three methodologies provide the basis for the UML 1.0.
- The classical 00 methodologies were used by numerous software development organizations and contributed to the bloom of the OO paradigm.
- But the software industry soon discovered that it was a nightmare to integrate and maintain systems that were developed using different methodologies.
- It was also very costly to support different tools that use different methodologies. These problems called for a unified modeling method and led to the creation of UML and UP.

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

https://en.wikipedia.org/wiki/Software_development_process
https://www.alliancesoftware.com.au/introduction-software-development-methodologies/

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

McGraw-Hill Education,'' Object-Oriented Software Engineering: An Agile Unified Methodology", 2013, Page No (21-39)

Course Faculty

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 : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **I** **Date of Lecture:**

Topic of Lecture: Software Process Models

Introduction : (Maximum 5 sentences) :

- Problems with the waterfall process have led researchers and funding agencies to find a better process that considers the wicked properties of software development.
- Tons of money and effort have been poured into this research. As a result, many software .process models have been proposed.
- Most models adopt an iterative, rather than a strictly sequential, process of activities.
- This section reviews some of these process models.

Prerequisite knowledge for Complete understanding and learning of Topic:

- software development
- Software Process and Methodology

Detailed content of the Lecture:

Prototyping Process

- The prototyping process model recognizes the mismatch between the- newly constructed software system and users' expectations, and the challenge to deliver the capabilities within the time and budget constraints.
- As a solution, a prototype of the system is constructed and-used to acquire and validate requirements.
- Prototypes are also used in feasibility studies as well as design validation.
- A prototype can be very simple or very sophisticated. A simple prototype shows only the look and feel and a sequence of screen shots to illustrate how the system would interact with a user.
- A sophisticated prototype may implement many of the system functions.
- Prototypes are generally classified into throwaway prototypes and evolutionary prototypes.
- A throwaway prototype is constructed quickly and economically-just enough to serve its purpose.
- A throwaway prototype could be reused in unit or integration testings as a reference implementation to check whether the implementation produces the correct result.

- Furthermore, it could be used to train users before the system IS released.

Evolutionary Process

- Prototypes help requirements acquisition, requirements validation, feasibility study, and validation of design ideas.
- However, throwaway prototypes imply that much effort is wasted. This is true when sophisticated prototypes are needed for feasibility study and design validation of large, real-time embedded systems.
- The evolutionary process model is aimed at solving this problem by letting the prototype evolves. It lets the users experiment with an initial prototype, constructed according to a set of preliminary requirements.

Personal Software Process

- The personal software process (PSP) is a comprehensive framework that is designed to train individual software engineers to improve their personal software processes.
- PSP consists of a series of scripts, forms, standards, and guidelines that the software engineer can apply to carry out a number of predefined programming exercises.

The PSP Process/Evolution

- To facilitate learning, the PSP uses an evolutionary approach. That is, the framework is presented in a series of predefined processes, named PSPO, PSPO.I, PSP1, PSPl.l, PSP2, PSP2.1 and PSP3.0.
- Each of these processes introduces a couple of good software engineering techniques or practices.

PSPO and PSPO.l.

- These two processes introduce process discipline and measurement.
- In particular, PSPO introduces the baseline process, time recording, defect recording, and defect type standard. PSPO.I introduces ceding standard, size measurement, and process improvement proposal.

PSP1 and PSP1.1.

- These two processes introduce estimations and planning.
- In particular, PSPI introduces size estimation and test report while PSPI.I covers planning and scheduling.

PSP2 and PSP2.1.

- These two processes introduce quality management and design.
- In particular, PSP2 presents code review and design review; and PSP2.1 introduces and design template.

PSP3.0.

- This process is designed to guide the development of component-level programs.

PSP Script

- In PSP, all processes are described using *process scripts* or scripts for short.
- Each script specifies the purpose, the entry criteria, the steps or activities of the process, and the exit criteria.

Spiral Process

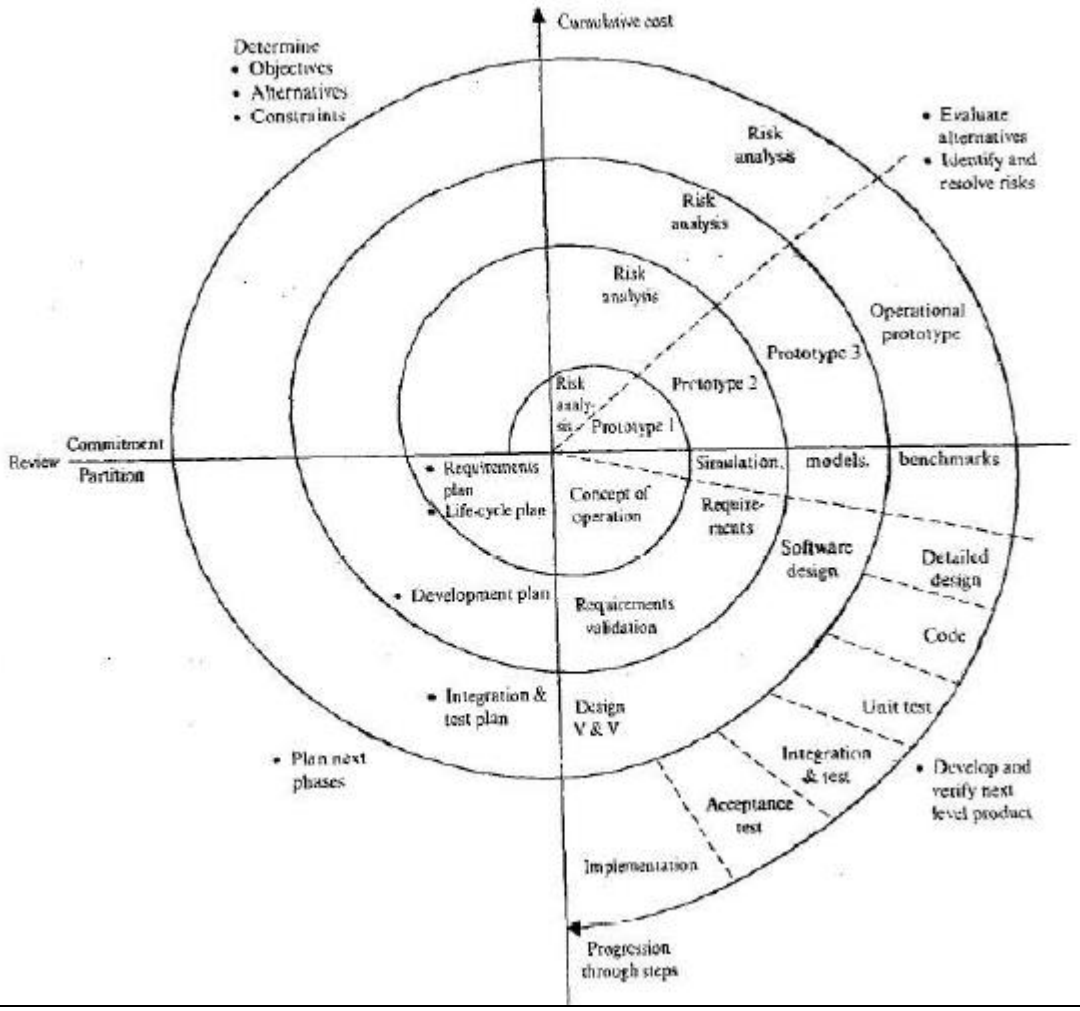
- The spiral process proposed by Barry, Boehm is known for its unique feature for risk management.
- Each cycle of the spiral is aimed at enhancing a certain aspect of the system under development.
- For example, functionality, performance or quality.

1. Determine the objectives alternatives, and constraints/or the current cycle (the Northwest corner of the spiral).

2. Evaluate alternatives, identify and evolve risks (the northeast corner of the spiral).

3.Develop and verify next level system

4. Plan next phases



Video Content / Details of website for further learning (if any):
http://users.csc.calpoly.edu/~jdalbey/308/Lectures/SoftwareProcessModels.html
https://www.youtube.com/watch?v=kwsKr1MObxs

Important Books/Journals for further learning including the page nos.:
McGraw-Hill Education,'' Object-Oriented Software Engineering: An Agile Unified Methodology", 2013,Page No (21-39)

Course Faculty

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

L-7

LECTURE HANDOUTS

CSE | **IV/II**

Course Name with Code : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **I** **Date of Lecture:**

Topic of Lecture: AGILE METHODS

Introduction : (Maximum 5 sentences) :

- Agile processes emphasize short iterations and frequent delivery of small increments.
- Although they differ in the naming and detail of the phases, all agile methods more or less cover requirements, design, implementation, integration, testing, and deployment activities during each iteration.

Prerequisite knowledge for Complete understanding and learning of Topic:

Detailed content of the Lecture:

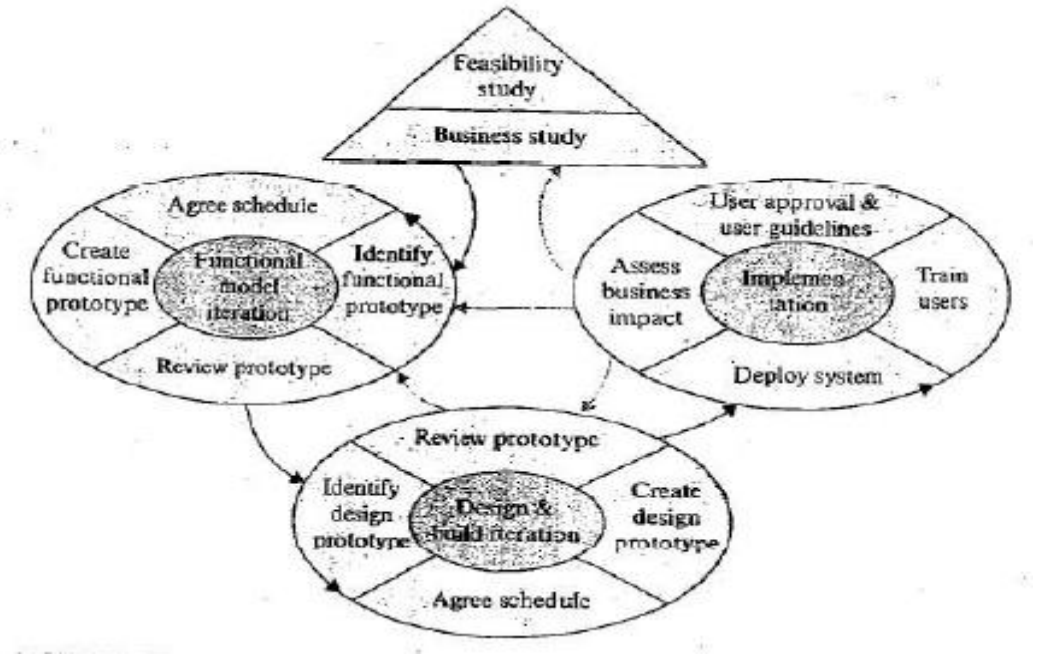
Agile Methods

- Like the evolutionary prototyping model and the spiral model, all agile methods adopt an iterative, incremental development process.
- However, all agile methods follow the agile manifesto presented.
- However, their emphases are different from conventional processes. For example, agile processes value working software over comprehensive documentation.
- This means barely enough modeling in the requirements and design phases.
- This section describes several of the most widely used agile methods.
- Each of these methods has a long list of principles, features, values and best practices.

Dynamic Systems Development

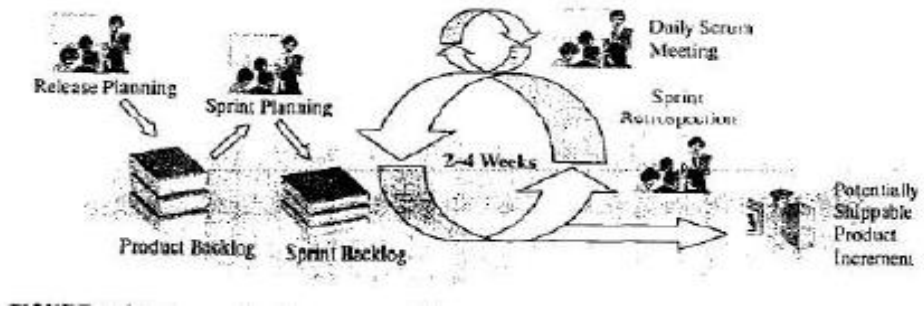
- The DSDM emerged in the early 19905 in the United Kingdom as an alternative to rapid application development (RAD). It is a process framework that different projects can adapt to

perform rapid application development. It has been deemed by some authors to be most suited to financial services applications.



Serum

- Scrum is a framework that allows organizations to employ and improve their software development practices.
- It consists of the Serum teams, the roles within a team, the time boxes, the artifacts, and the Serum rules.
- Scrum is an iterative, incremental approach that aims to optimize predictability and control risk.



Feature Driven Development

- The Feature Driven Development (FDD) method consists of six steps or phases.
- The first three are performed once and the last three are iterative.
- The FDD method is considered more suitable for developing mission critical systems by its advocates.
- The six phases of FDD are briefly described as follows:

1. **Develop overall model.**
    - During this phase, a domain expert provides a walkthrough of the overall system, which may include decomposition into subsystems and components.

- Additional walkthroughs of the subsystems or components may be provided by experts in their domains. Based on the walkthroughs, small groups of developers produce object models for the respective domains.
- The development teams then work together to produce an overall model for the system.

2. **Build a feature list.**
   - During this phase, the team produces a feature list representing the business functions to be delivered by the system.
3. **Plan by feature.**
   - During this phase, the team produces an overall plan to guide the incremental development and deployment of the features, according to their priorities and dependencies. The features are assigned to the chief programmers.
4. **Design by feature; build by feature, and deployment.**
   - These three phases are iterative, during which the increments are designed, implemented, reviewed, tested, and deployed. Multiple teams may work on different sets of features simultaneously.
   - Each increment lasts a few days to a few weeks.

Extreme Programming

- Extreme programming or XP is an agile method suitable for small teams facing vague and changing requirements.
- The XP process consists of six phases:
  1. Exploration.
  2. Planning.
  3. Iterations to first release.
  4. Productionizing
  5. Maintenance
  6. Death

Video Content / Details of website for further learning (if any):
https://resources.collab.net/agile-101/agile-methodologies
https://www.youtube.com/watch?v=ZZ_vnqvW4DQ

Important Books/Journals for further learning including the page nos.:
McGraw-Hill Education,'' Object-Oriented Software Engineering: An Agile Unified Methodology", 2013, Page No (40-44)

Course Faculty

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

L-8

LECTURE HANDOUTS

CSE | **II/IV**

Course Name with Code : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **I** **Date of Lecture:**

Topic of Lecture: SYSTEM ENGINEERING REQUIREMENTS

Introduction : (Maximum 5 sentences) :

- System engineering is a multidisciplinary approach to develop systems that involve hardware, software, and human components.
- System engineering defines the system requirements and constraints for the system.
- It allocates the requirements to the hardware, software, and human subsystems, and integrates these subsystems to form the system.
- Software engineering is a part of system engineering.
- Many systems are embedded systems.
- An embedded system consists of hardware, software, and human components. These components interact with each other to accomplish the mission of the system.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Software engineering
- Software Development

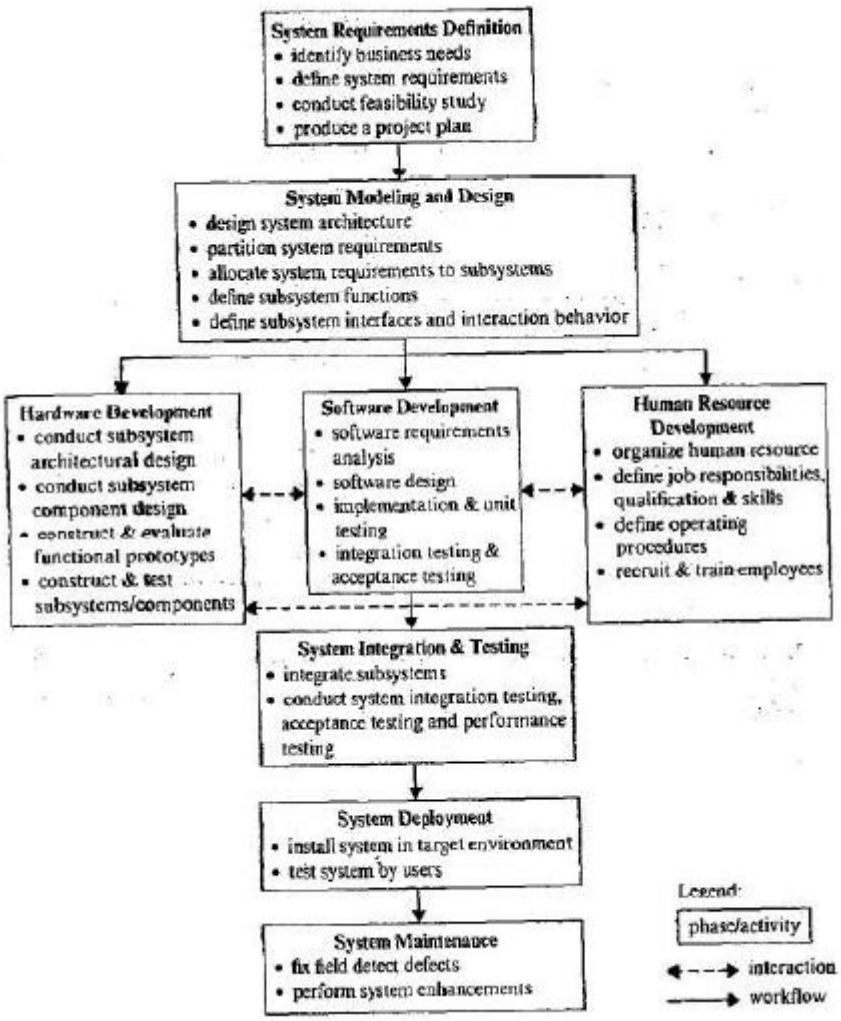
Detailed content of the Lecture:

WHAT IS A SYSTEM?

- A system consists of components that interact with each other to accomplish a purpose.
- A system can be big or small, complex or simple, and exist physically or only conceptually.
- For example, the universe is a very large system that has been in existence for millions of years. An ant is a very small system. These systems are natural systems.
- In contrast to natural systems, there are many man-made systems. Man-made systems may exist physically or only conceptually. Mathematical logic, number systems, measurement systems, and many classification systems are examples of conceptual systems.

SYSTEM ENGINEERING

- System development for the ABHS must consider the total system rather than the software system alone.
- In addition, the ABHS involves multiple engineering disciplines including electrical and electronic engineering, mechanical engineering, civil engineering, and software engineering.



SYSTEM REQUIREMENTS DEFINITION

- System requirements definition identifies the business needs and specifies the system requirements.
- It begins with an initial system concept and expands and refines the concept. During this process, a set of capabilities that the system must deliver is identified.
- These capabilities are formulated as system requirements. The system requirements include functional requirements, quality requirements, performance requirements, and other system-specific requirements. This section describes the system requirements definition activity.

1. **Identifying Business Needs**
    - Identifying business needs begins with an information collection activity.
    - That is, information about the business goals and the current business situation is collected.
    - The team identifies the gap between the current situation and the business goals, and derives the business needs.

The information collection activity answers the following questions:

1. What is the business that-the system will automate?
2. What is the system's environment or context?
3. What are the business goals or product goals?
4. What is the current business situation, and how does it operate?
5. What are the existing business processes, and how do they relate to-each other'!
6. What are the problems with the current system?
7. Who are the users of the current system and the future system, respectively?
8. What do the customer and users want from the future system, and what are their business priorities?
9. What are the quality, performance, and security considerations?

Defining System Requirements

- The next step is deriving system requirements from the business needs identified.
- For example, the capabilities of the ABHS are derived to satisfy the needs of the ABHS.
- However, not all needs are to be satisfied due to budget, delivery schedule technology, and political constraints as well as cost-effectiveness considerations.

The requirements are numbered to facilitate reference:

Rl. ABHS shall check in and transport luggage to departure gates and baggage claim areas according to the destinations of the passengers.

R2. ABHS shall allow airline agents to inquire about luggage status and to locate luggage.

R3. ABHS shall check all baggage and detect items that are prohibited.

R4. ABHS shall be able to serve 20,000 passengers per day.

Video Content / Details of website for further learning (if any):
https://www.sebokwiki.org/wiki/System_Requirements
https://www.youtube.com/watch?v=qaiSB1bdS_8

Important Books/Journals for further learning including the page nos.:
McGraw-Hill Education,'' Object-Oriented Software Engineering: An Agile Unified Methodology", 2013, Page No (53-60)

Course Faculty

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 : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **I** **Date of Lecture:**

Topic of Lecture: Architectural Design

Introduction : (Maximum 5 sentences) :

- After the system requirements are identified, the next logical step is to design the
- System to satisfy the system requirements. Ideally, the system should be designed and
- Implemented by engineers who are experts in all the engineering disciplines involved.
- Unfortunately, such engineers are hard to find and expensive to hire. Therefore, systems
- Are usually decomposed into a hierarchy of subsystems. which can be developed

Prerequisite knowledge for Complete understanding and learning of Topic:

- system requirements

Detailed content of the Lecture:

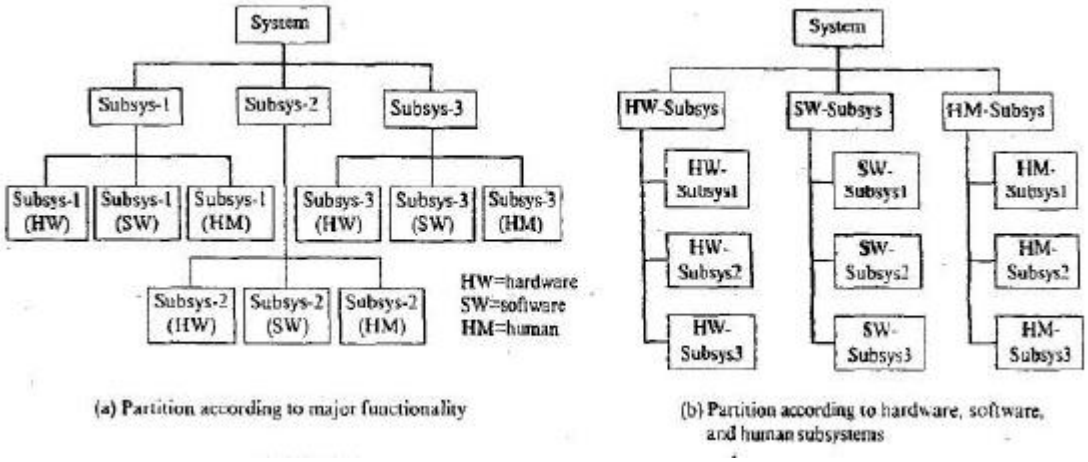
SYSTEM ARCHITECTURAL DESIGN

System architectural design performs the following interrelated activities:

1. Decompose the system into a hierarchy of subsystems.
2. Allocate system requirements to subsystems.
3. Visualize the system architecture.

System Decomposition

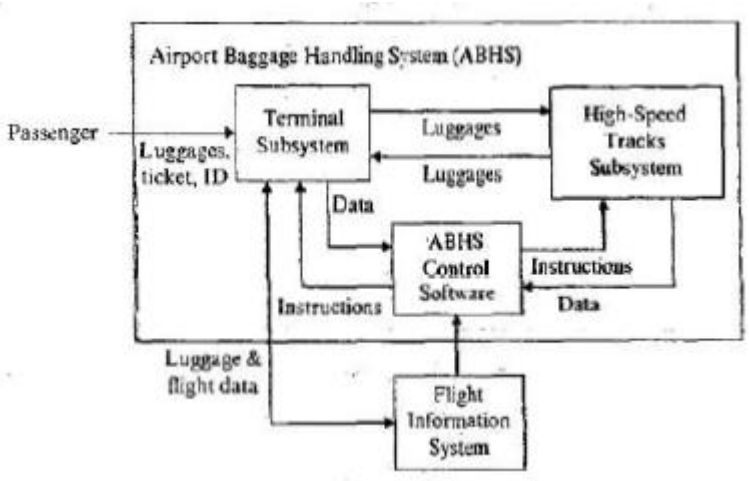
- One important task of system architectural design is identifying the subsystems of the system. A top-down, divide-and-conquer approach is often used.
- In particular the approach decomposes the system into a hierarchy of subsystems. This approach reduces the complexity of system development because each subsystem is easier to design and implement.



- There are different ways to decompose a system.
- Therefore, the result is not unique. System decomposition aims at accomplishing the following goals:
    1. The result should enable separate engineering teams to develop the subsystems.
    2. The result should facilitate the use of commercial off-the-shelf (COTS) parts.
    3. The result should partition or nearly partition the system requirements.
    4. Each subsystem should have a well-defined functionality.
    5. The subsystems should be relatively independent.
    6. The subsystems should be easy to integrate.

Architectural Design Diagrams

- It is a common practice to construct application models and system models during system design.
- These models help the team understand and analyze the application domain, business processes, and workflows to identify problems, and develop and evaluate design solutions.
- Various diagrams are used to depict different aspects of the application and the system. Block diagrams, Unified Modeling Language (UML), and its extension, System Modeling Language (SysML), and data flow diagrams are widely used during system modeling and design.



Block diagram for an airport baggage handling system

- The ability of UML to support system modeling leads to an extension of UML,that is, the System Modeling Language (SysML). The nine diagrams of SysML and how they relate to UML are summarized in below diagram.

SysML	UML	Description	Remark	Presented in Chapter
Activity diagram	Activity diagram	Model activities that relate to each other via workflows, and exhibit sequencing, exclusion, synchronization, and concurrency relationships	SysML activity diagram extends UML activity diagram.	14
Block definition diagram		Model structural elements called blocks and their composition and classification	Block definition diagram extends UML class diagram.	
Internal block diagram		Model interconnection and interfacing of internal elements of a block	Internal block diagram extends UML composite structure diagram.	
Package diagram	Package diagram	Model the logical organization of modeling artifacts and software artifacts	Same diagrams	
Parametric diagram		Specifies constraints to support engineering analysis		
Requirement diagram		Model text-based requirements and their relationships with other requirements and artifacts such as design elements, test cases, etc.		
Sequence diagram	Sequence diagram	Model time-ordered interaction behavior between objects	Same diagrams	9
State diagram	State diagram	Model state dependent behavior of an object	Same diagrams	13
Use case diagram	Use case diagram	Show the functions or business processes of an application or system as well as relationships of these to external entities called actors	Same diagrams	7

Specification of Subsystem Functions and Interfaces

- This step specifies the functionality of each subsystem and how the subsystems interact with each other.
- The functionality is specified according to the system requirements allocated to the subsystem. It refines the requirements assigned to each subsystem.
- The interfaces between the subsystems specify how the subsystems connect and communicate with each other.
- The interaction behavior specifies the sequences of messages exchanged between the subsystems.
- These enable the teams that implement the subsystems to know what interfaces and interaction behavior they can expect and need to provide.

Video Content / Details of website for further learning (if any):
http://ecomputernotes.com/software-engineering/architecturaldesign
https://www.youtube.com/watch?v=ly8orBNiNQM

Important Books/Journals for further learning including the page nos.:
McGraw-Hill Education,'' Object-Oriented Software Engineering: An Agile Unified Methodology", 2013, Page No (61-72)

Course Faculty

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-

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code :Object Oriented Software Engineering/16CSD04

Course Faculty :

Unit : IIDate of Lecture:

Topic of Lecture:SOFTWARE REQUIREMENTS ELICITATION

Introduction :

- Software systems are built for many different reasons.
- A software project is successful if the system satisfies its software requirements, the budget is not overrun, and the system is delivered as scheduled.
- The main difference between requirements and constraints is that constraints reduce the number of design and implementation alternatives.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- Software Engineering
- Software development

Detailed content of the Lecture:

IMPORTANCE OF REQUIREMENTS ELICITATION

- Two real-world stories illustrate the importance of requirements elicitation. More than 40 years ago in the beginning of the 1970s. I had the opportunity to work on a project for the electric utility industry. We worked days and nights for two years, meeting the customer representatives Performing design, implementation, and testing. Finally we delivered the system to the customer and celebrated the victory with a champagne party.

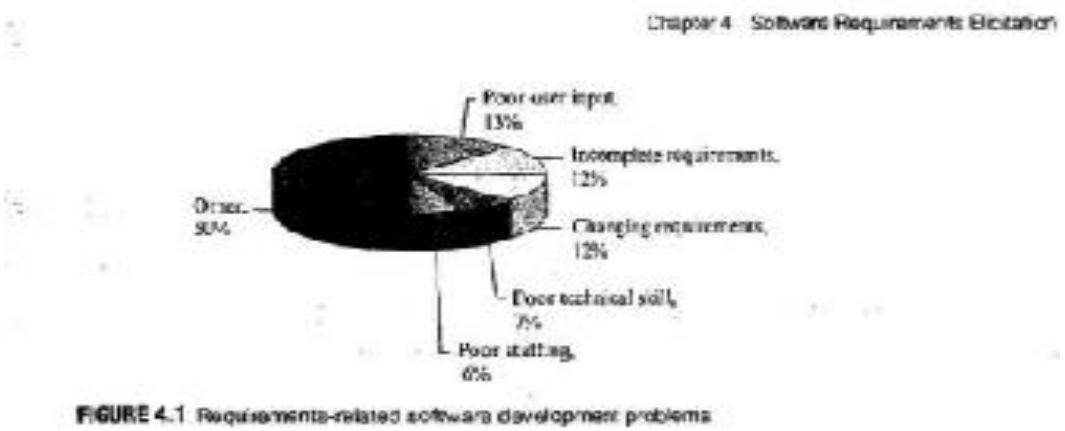


FIGURE 4.1 Requirements-related software development problems

CHALLENGES OF REQUIREMENTS ELICITATION:

- The development team does not know enough about the application and application domain.
- Customers and users do not know what software can do and how to express their needs.

- Lack of a common background creates a communication barrier between the team and customer and users.

TYPES OF REQUIREMENT

- Performance requirements
- Quality requirements
- Safety requirements
- Security requirements
- Interface requirements

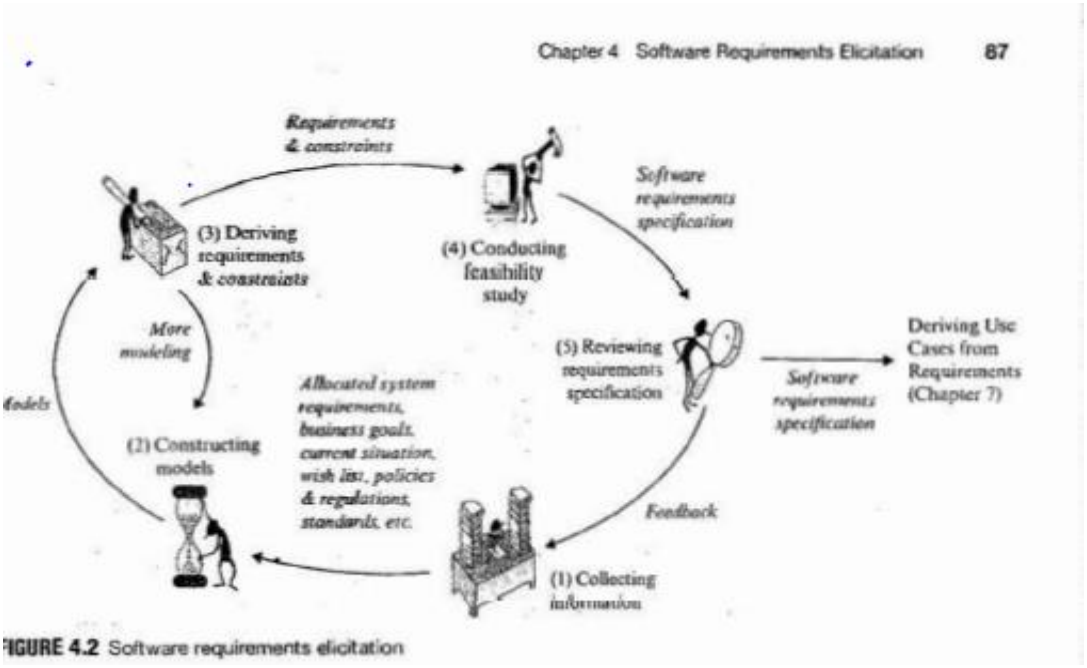


FIGURE 4.2 Software requirements elicitation

STEPS FOR REQUIREMENTS ELICITATION

Step 1. Collecting information about the application.
Step 2. Constructing analysis models if desired.
Step 3. Deriving requirements and constraints.
Step 4. Conducting feasibility study.
Step 5. Reviewing the requirements specification.

Focuses of Information-Collection Activities

The information-collection activities must focus on acquiring information about the application the business processes, and the application domain.

Information-Collection Techniques

Information-collection methods and techniques are applied to find answers to the questions presented previously. These techniques include

1. Customer presentation
2. Literature survey
3. Study of existing business procedures and forms
4. Stakeholder survey
5. User interviewing
6. Writing user stories

Current Business Situation

- The OIE is located in a building somewhat distant from the main campus. It is difficult for students to access the OIE.
- The Study Abroad Program of the OIE is mainly a manual operation. It is time consuming to process student inquiries and study abroad applications.

Business Goals

1. Greatly facilitate students' access to the OIE Study Abroad Program.

2. Significantly improve the effectiveness and efficiency of the services provided by the Study Abroad Program.

Wish List

A website for the Study Abroad Program. The system is named Study Abroad Management System (SAMS). A list of similar websites was suggested by the OIE.

REQUIREMENTS MANAGEMENT AND TOOLS

Requirements change is common for many software projects. The changes include adding new requirements, modifying and deleting existing requirements.

Changing a requirement may affect several artifacts and other requirement items.

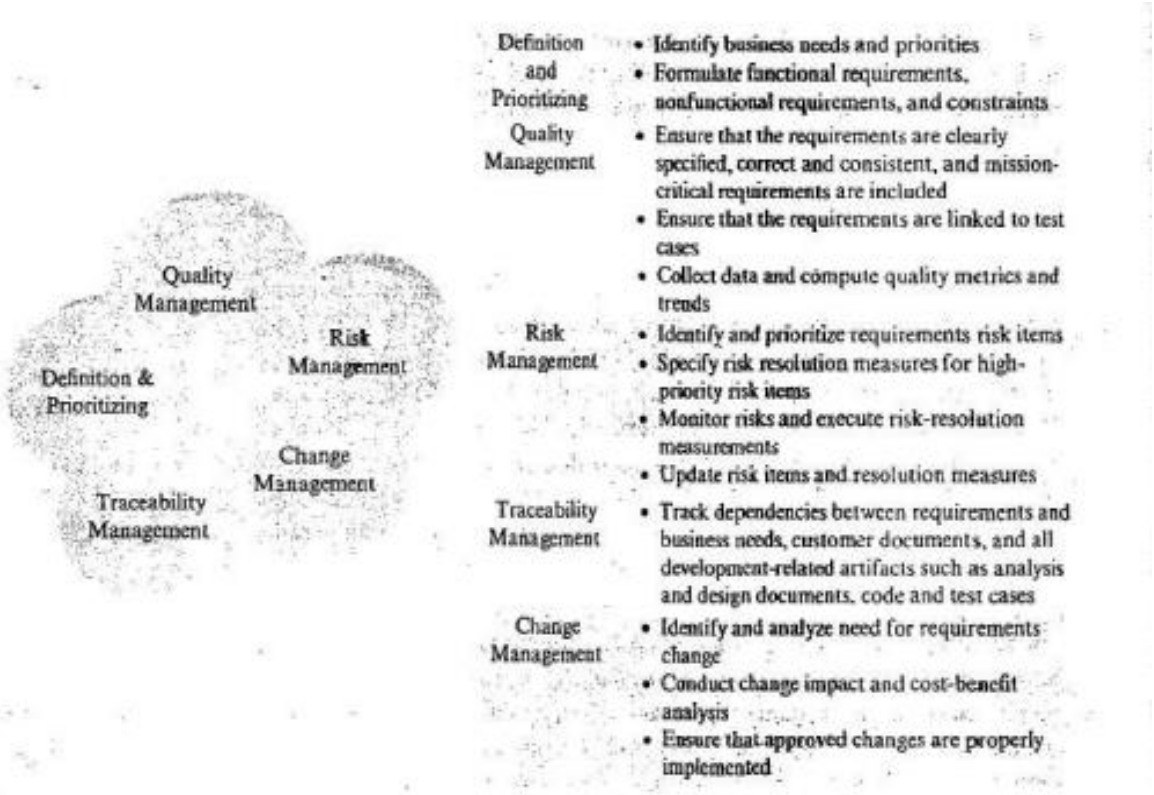


FIGURE 4.6 Requirements management activities

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

https://www.geeksforgeeks.org/software-engineering-requirements-elicitation/
https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=12015
https://www.youtube.com/watch?v=42_J-W7RumE

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

Books

1. David Kung Object-Oriented Software Engineering:An Agile Unified Methodology McGraw-Hill Education 2013 Page No:80-101

Journals

https://www.researchgate.net/publication/326544173
https://ieeexplore.ieee.org/document/8513829

Course Faculty

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-

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code :Object Oriented Software Engineering/16CSD04

Course Faculty :

Unit :IIDate of Lecture:

Topic of Lecture:DOMAIN MODELING

Introduction : (Maximum 5 sentences) :

- Domain modeling is a conceptualization process to help the development team understand the application domain.
- Five easy steps:
    - collecting information about the application domain;
    - brainstorming;
    - classifying brainstorming results;
    - visualizing the domain model using a UML class diagram;
    - Performing-inspection and review.

Prerequisite knowledge for Complete understanding and learning of Topic:
Software

Detailed content of the Lecture:

One of the tools that enables the engineer to this is called domain modeling. Throughout this chapter. you will learn:

- Domain modeling.
- The importance of domain modeling.
- Object-oriented concepts.
- Domain modeling steps.
- UML class diagram.

Domain Modeling:

- Domain modeling is a conceptualization process. It aims to identify. important domain concepts, their properties, and relationships between the concepts. The result is portrayed in a diagram called a domain model.
- A software engineer like Mary, who works in IT consulting, of tell works on projects in different application domains.
- Even if a software engineer is not a consultant. He or she may be required to work on new projects or novel extensions of existing systems.
- Software engineering is both challenging and full of' excitement because engineers get to work on new applications from time to time.
- A good software engineer can quickly understand anew application domain.

IMPORATNCE:

- The construction of a domain model helps in identifying and resolving differences in perception. In particular.
- Domain modeling helps the development team or the analyst understand the application and the application domain.
- Domain modeling Jets the team members communicate and improve their common perception of the application and application domain.
- Domain modeling helps the development team communicate their perception to the customer or users and seek feedback.
- Domain modeling provides a common conceptual basis for the subsequent design, implementation, testing, and maintenance.
- A domain model can help new team members understand the relevant application and the application domain.
- The conceptualization process involves observation, classification, abstraction, and generalization. The process is important because in a broad sense software is merely a conceptual product.
- This requires the development team to understand the entities or objects in the banking application and how they relate to each other, the properties or states of the banking objects and so on.
- If the differences in perception are immaterial, then they will not significantly impact the design, implementation, integration, testing, and maintenance of the software product.

Object Oriented Concepts

Itstarted right from the moment computers were invented. Programming was there, and programming approaches came into the picture.

Programming is basically giving certain instructions to the computer.

Domain Modeling

It is understood as abstract modeling. a site model could be an illustration of the ideas or objects showing within the drawback domain. It additionally captures the apparent relationships among these objects. Samples of such abstract objects area unit the Book, Book Register, member register, Library Member, etc.

- Boundary Objects
- Entity Objects
- Controller Objects

UML Class Diagram

It is the general purpose modeling language used to visualize the system. It is a graphical language that is standard to the software industry for specifying, visualizing, constructing and documenting the artifacts of the software systems, as well as for business modeling.

Benefits of UML:

- Simplifies complex software design, can also implement OOPs like concept which is widely used.
- It reduces thousands of words of explanation in a few graphical diagrams that may reduce time consumption to understand.
- It makes communication more clear and real.
- It helps to acquire the entire system in a view.
- It becomes very much easy for the software programmer to implement the actual demand once they have the clear picture of the problem.

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

https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/
https://www.tutorialride.com/software-engineering/oo-design-concept-in-software-engineering.htm
https://www.geeksforgeeks.org/software-engineering-domain-modeling/
https://www.youtube.com/watch?v=UI6lqHOVHic

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

BOOKS:

David Kung Object-Oriented Software Engineering:An Agile Unified Methodology McGraw-Hill Education 2013 Page No:105-107

Journals

https://link.springer.com/article/10.1007/BF02687879
https://www.researchgate.net/publication/324345033_Optimizing_UML_Class_Diagrams

Course Faculty

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

L-

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code :**Object Oriented Software Engineering/16CSD04**

Course Faculty :

Unit : **IIDate of Lecture:**

Topic of Lecture:OBJECT-ORIENTATION

Introduction : (Maximum 5 sentences)

- Domain modeling aims to identify important domain concepts and their properties, and relationships between the concepts.
- These are represented as classes and relationships between the classes, and can be depicted with UML class diagrams.
- This section reviews some of the basic concepts of the object-oriented paradigm and how these are displayed in UML class diagrams.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

Detailed content of the Lecture:

Extensional and Intentional Definitions:

An extensional definition defines a concept by enumerating instances of the concept.

An extensional definition of even numbers could be the set of numbers consisting of ... , -4, - 2, 0: 2, 4, ...

For the child, the extensional definition of "dog" consists of his neighbor's dog. the dog across the street, the dog that lives near the park, and so forth.

As the child sees more of the world, he learns that cats and dogs behave differently but that tile are animals.

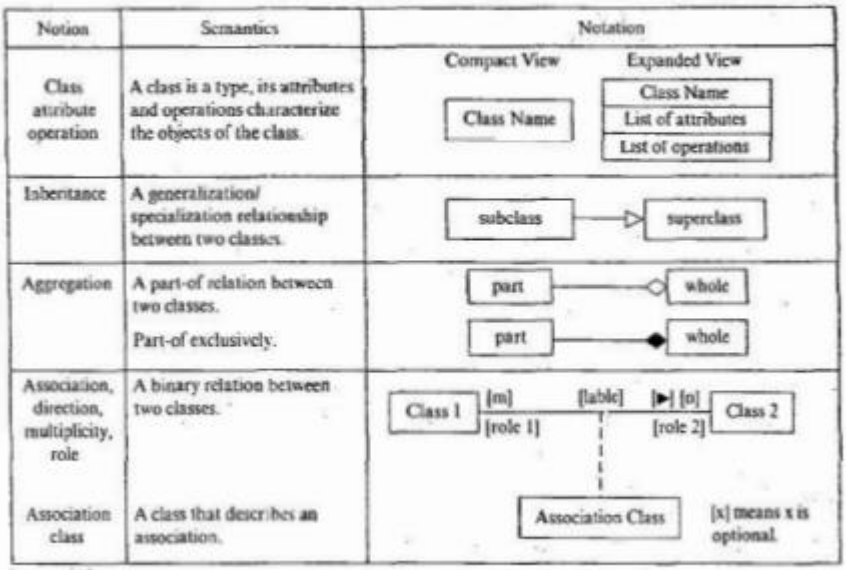
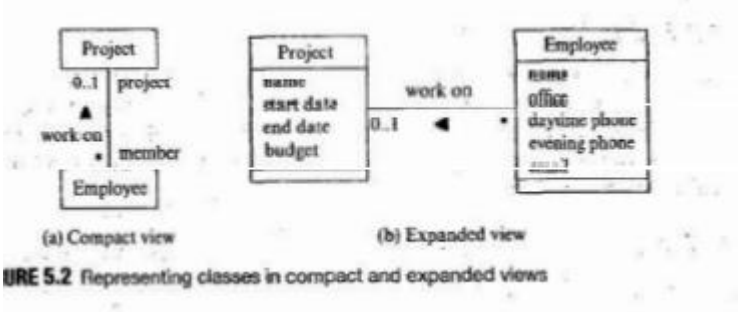


FIGURE 5.1 Some commonly used class diagram notions and notations

Class and Object

A class is a type, an intentional definition ora concept. A class encapsulates its attributes and operations that characterize the instances of the class. An object is an instance of a class.



URE 5.2 Representing classes in compact and expanded views

Object and Attribute

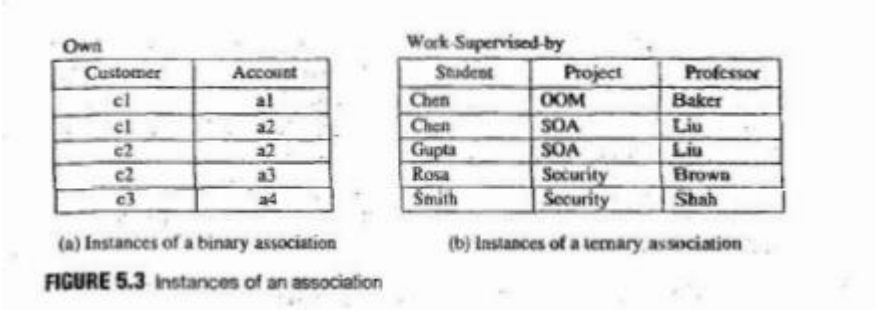
An application domain object has an independent existence in the application or application domain; an attribute does not.

Attribute describe and characterize object.

Attributes can be entered from an input device but objects cannot: Objects are created by calling a function.

ASSOCIATION

An association is a relation between one or more classes. It states that 0njects of one class may relate to objects of the other classes.



Own

Customer	Account
c1	a1
c1	a2
c2	a2
c2	a3
c3	a4

(a) Instances of a binary association

Work-Supervised-by

Student	Project	Professor
Chen	OOM	Baker
Chen	SOA	Liu
Gupta	SOA	Liu
Rosa	Security	Brown
Smith	Security	Shah

(b) Instances of a ternary association

FIGURE 5.3 Instances of an association

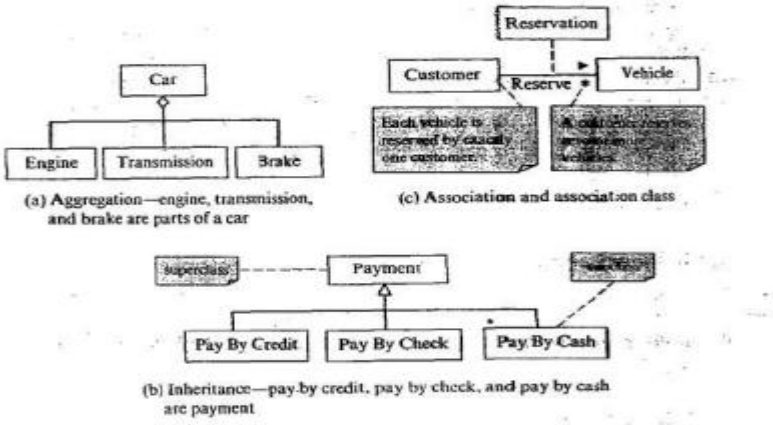
Multiplicity and Role

The multiplicity of a class with respect to an association is an assertion to the number of instances of the class that may relate each combination of one instance of each of other classes in the association.

0..1	zero or one	m..n	m to n
0..m	zero to m	m..*	m or more
,0..	zero or more	m	exactly m
1	exactly one (default)	1..*	one or more
i,j,k	explicitly enumerated		

FIGURE 5.4 Symbols for expressing various multiplicity assertions

Aggregation



(a) Aggregation—engine, transmission, and brake are parts of a car

(c) Association and association class

(b) Inheritance—pay by credit, pay by check, and pay by cash are payment

FIGURE 5.5 Representing relationship and association class

Inheritance:

Inheritance is a binary relation between two concepts or classes such that one concept or class is a generalization of the other.

Association Class

An Association class is a special class that defines properties and behaviors for the instances of an association.

Student	Course	Semester	Grade
Bachman	AI	Spr 07	A
Backman	DB	Spr 07	A
Backman	SE	Spr 07	A
Chang	AI	Spr 07	B
Chang	DB	Spr 07	A
Chang	SE	Spr 07	A
Chang	Compiler	Fal 06	B
Chang	Algorithms	Fal 06	A
Chang	Programming	Fal 06	B

FIGURE 5.6 Tabular representation of objects of an association class

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

https://www.tutorialspoint.com/uml/uml_class_diagram.htm
https://www.youtube.com/watch?v=o_-1HSAaWTQ

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

Books

2. David Kung Object-Oriented Software Engineering:An Agile Unified Methodology McGraw-Hill Education 2013 Page No:107-117

Journals

https://www.researchgate.net/publication/220625913_Role_of_UML_Class_Diagram_in_Object-Oriented_Software_Development
http://downloads.hindawi.com/journals/sp/2015/421816.pdf

Course Faculty

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

L-

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code :**Object Oriented Software Engineering/16CSD04**

Course Faculty :

Unit : **IIDate of Lecture:**

Topic of Lecture:OBJECT-ORIENTATION -CLASS DIAGRAM

Introduction : (Maximum 5 sentences)

- Domain modeling aims to identify important domain concepts and their properties, and relationships between the concepts.
- These are represented as classes and relationships between the classes, and can be depicted with UML class diagrams.
- This section reviews some of the basic concepts of the object-oriented paradigm and how these are displayed in UML class diagrams.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

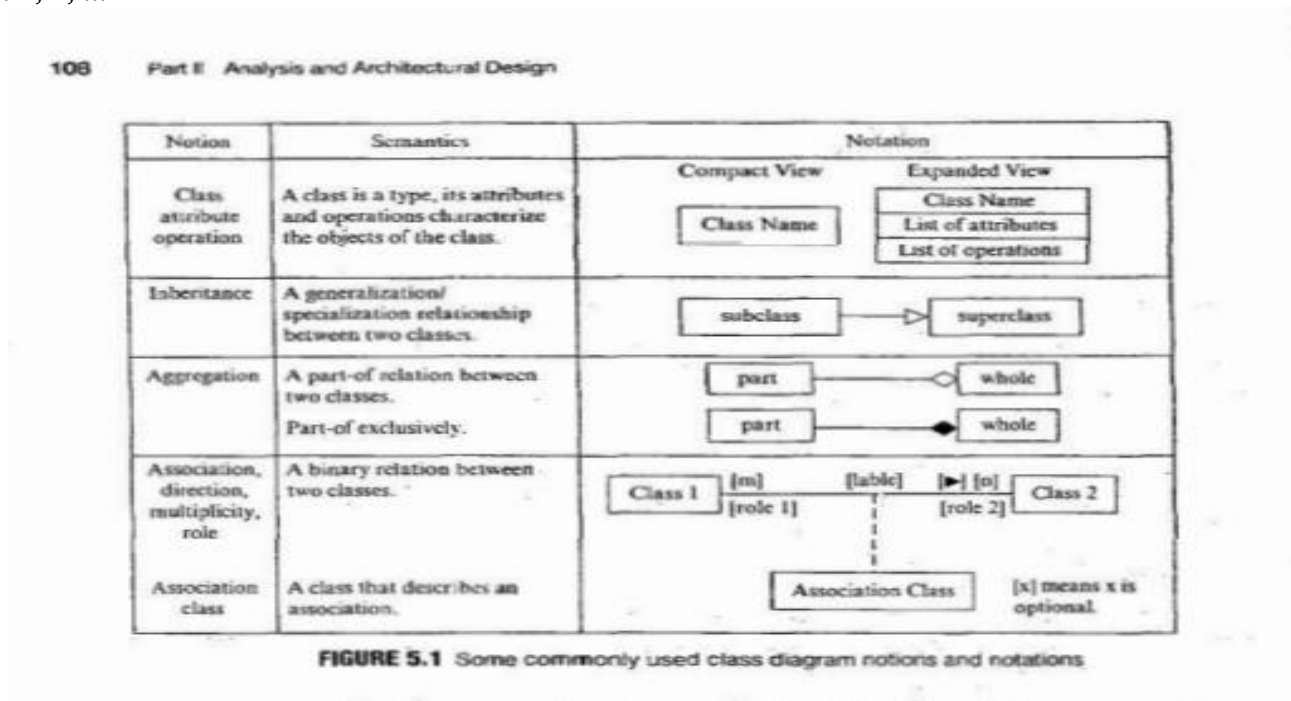
- Software Requirements
- Domain Modeling

Detailed content of the Lecture:

Extensional and Intentional Definitions:

An extensional definition defines a concept by enumerating instances of the concept.

An extensional definition of even numbers could be the set of numbers consisting of ... , -4, - 2, 0: 2, 4, ...



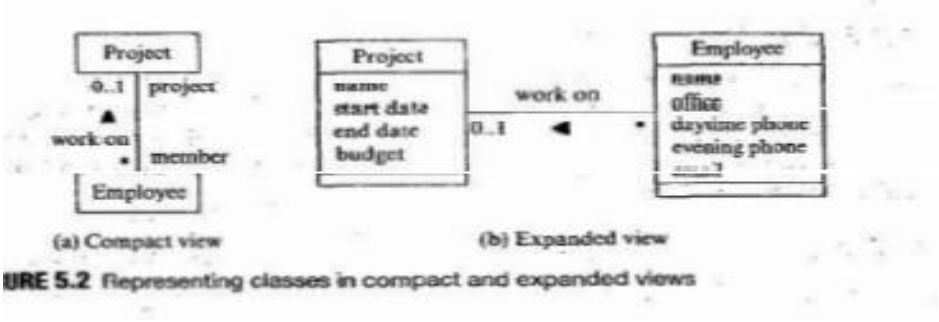
108 Part II Analysis and Architectural Design

Notion	Semantics	Notation
Class attribute operation	A class is a type, its attributes and operations characterize the objects of the class.	Compact View: Class Name. Expanded View: Class Name / List of attributes / List of operations
Inheritance	A generalization/ specialization relationship between two classes.	subclass → superclass
Aggregation	A part-of relation between two classes.	part ◇— whole
	Part-of exclusively.	part ◆— whole
Association, direction, multiplicity, role	A binary relation between two classes.	Class 1 [m] [role 1] [lable] [►] [n] [role 2] Class 2
Association class	A class that describes an association.	Association Class; [x] means x is optional.

FIGURE 5.1 Some commonly used class diagram notions and notations

Class and Object

A class is a type, an intentional definition ora concept. A class encapsulates its attributes and operations that characterize the instances of the class. An object is an instance of a class.



URE 5.2 Representing classes in compact and expanded views

Object and Attribute

An application domain object has an independent existence in the application or application domain; an attribute does not.

Attribute describe and characterize object.

Attributes can be entered from an input device but objects cannot: Objects are created by calling a function.

ASSOCIATION

An association is a relation between one or more classes. It states that 0njects of one class may relate to objects of the other classes.

Own

Customer	Account
c1	a1
c1	a2
c2	a2
c2	a3
c3	a4

(a) Instances of a binary association

Work Supervised-by

Student	Project	Professor
Chen	OOM	Baker
Chen	SOA	Liu
Gupta	SOA	Liu
Rosa	Security	Brown
Smith	Security	Shah

(b) Instances of a ternary association

FIGURE 5.3 Instances of an association

Multiplicity and Role

The multiplicity of a class with respect to an association is an assertion to the number of instances of the class that may relate each combination of one instance of each of other classes in the association.

0..1	zero or one	m..n	m to n
0..m	zero to m	m..*	m or more
,0..	zero or more	m	exactly m
1	exactly one (default)	1..*	one or more
i,j,k	explicitly enumerated		

FIGURE 5.4 Symbols for expressing various multiplicity assertions

Aggregation

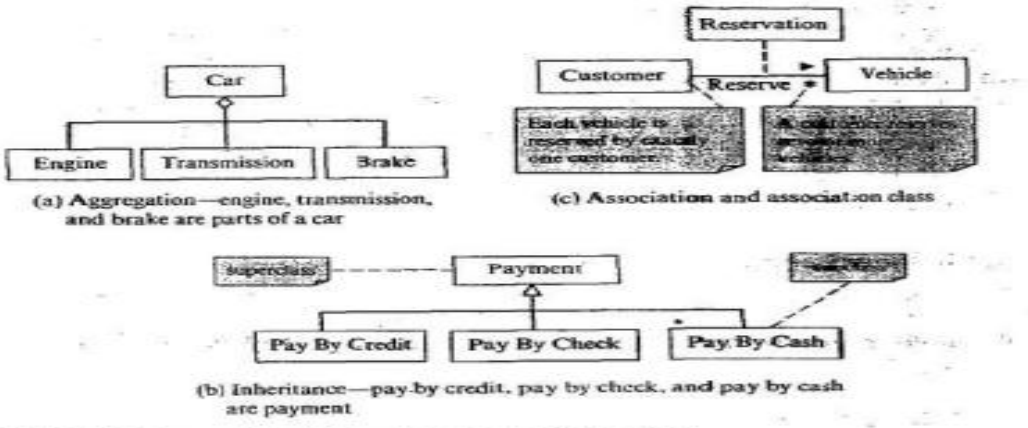


FIGURE 5.5 Representing relationship and association class

Inheritance:

Inheritance is a binary relation between two concepts or classes such that one concept or class is a generalization of the other.

Association Class

An Association class is a special class that defines properties and behaviors for the instances of an association.

Student	Course	Semester	Grade
Bachman	AI	Spr 07	A
Backman	DB	Spr 07	A
Backman	SE	Spr 07	A
Chang	AI	Spr 07	B
Chang	DB	Spr 07	A
Chang	SE	Spr 07	A
Chang	Compiler	Fal 06	B
Chang	Algorithms	Fal 06	A
Chang	Programming	Fal 06	B

FIGURE 5.6 Tabular representation of objects of an association class

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

https://www.tutorialspoint.com/uml/uml_class_diagram.htm
https://www.youtube.com/watch?v=o_-1HSAaWTQ

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

Books

3. David Kung Object-Oriented Software Engineering:An Agile Unified Methodology McGraw-Hill Education 2013 Page No:107-117

Journals

https://www.researchgate.net/publication/220625913_Role_of_UML_Class_Diagram_in_Object-Oriented_Software_Development
http://downloads.hindawi.com/journals/sp/2015/421816.pdf

Course Faculty

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

LECTURE HANDOUTS

CSE

II/IV

L-

Course Name with Code :**Object Oriented Software Engineering/16CSD04**

Course Faculty :

Unit : **IIDate of Lecture:**

Topic of Lecture:STEPS FOR DOMAIN MODELING

Introduction : (Maximum 5 sentences)

With the object-oriented notions reviewed in the previous sections, it is time to describe the steps for domain modeling. It shows the steps and their input and output.

These steps may need to be kilted a few times to produce a good domain model. They are outlined as follows and described in detail.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Domain Modeling
- Object Orientation

Detailed content of the Lecture:

Step 1. Collecting application domain information.

- The first step to domain modeling is collecting application domain information.
- Techniques for collecting application domain information have been described previously and will be reviewed-again.
- The output of this step includes all relevant information Documentation about the application.

Step 2. Brainstorming.

After collecting information about the application domain, the development team members meet together to identical! and list important application domain concepts as described in Section

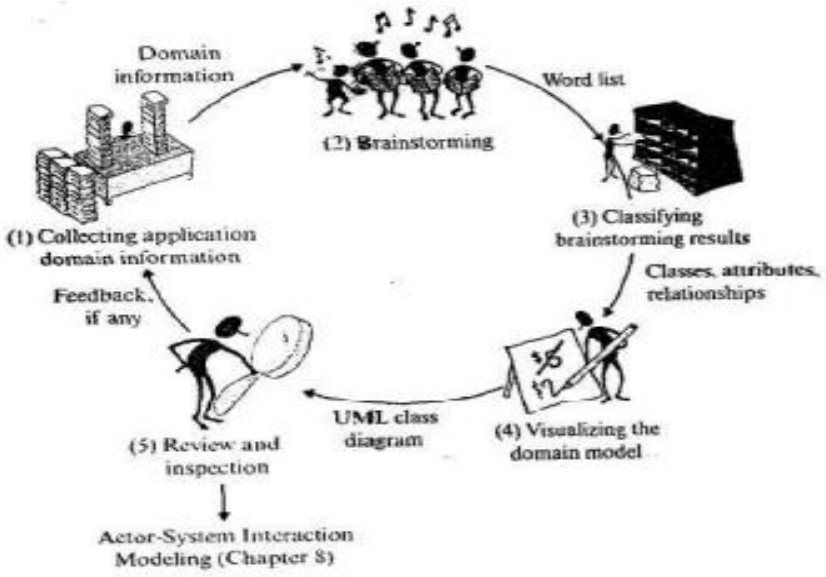


FIGURE 5.7 Steps for domain modeling

Collecting Application Domain Information:

- Customer presentation.
- Interviewing customer representatives, users, and domain experts.
- Study of relevant literature.
- Study of similar projects.
- Study of business documentation and forms.
- Study of government policies and regulations.
- Study of industry standards.
- Development and use of questionnaires.

Brainstorming

Focus on domain specific or domain relevant concepts and relationships.
Ignore design and implementation concepts.

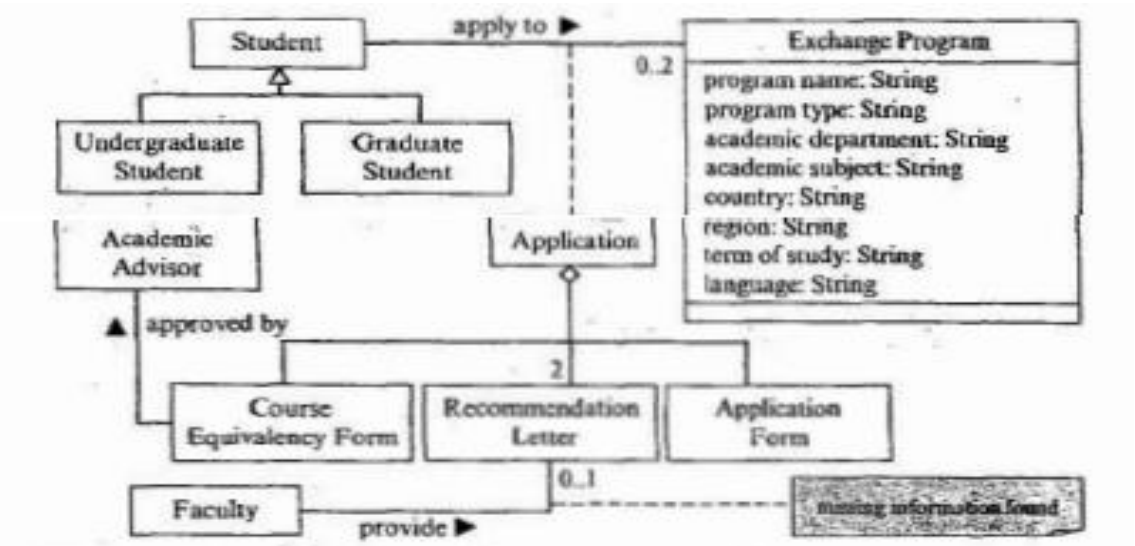


FIGURE 5.13 Converting classification result into a class diagram

Classifying Brainstorming Results

- The listed phrases ace classified into classes, attributes, attribute values, and relationships.
- This is done by applying the classification rules shown
- The classification codes in Figure 5.l0 are used to indicate the classification result

Rule #	Phrase Identified	Corresponding Modeling Concept
1	noun/noun phrase	
	(a) has independent existence	class
	(b) a role played by some object	role in association
	(c) describes a many-to-many relationship	association class
	(d) is generalization/specialization of	superclass/subclass
	(e) does not exist independently in the application/domain	attribute of some class
2	"X of Y" expression	
	(a) X exists independently in the application/domain	X is part-of Y or Y is an aggregation of X
	(b) X does not exist independently in application/domain	X is an attribute of Y
	(c) X denotes a role played by some object	X is a role in an association
3	transitive verb	association relationship
4	adjective/adverb/enumeration	attribute value
5	numeric	
	(a) relevant concept is an attribute	attribute value
	(b) relevant concept is an object	multiplicity
6	possession expression (e.g., Y has/have/possesses X, etc.)	
	(a) X has independent existence in application/domain	Y is an aggregation of X
	(b) otherwise	X is an attribute of Y
7	"consist of/part of/is composed of" expression	aggregation relationship
8	containment/containing expression	
	(a) contained object(s) can be removed without affecting integrity of containing object	association
	(b) otherwise	aggregation
9	"X is Y" or generalization/specialization expression	inheritance

Note: (1) X is an attribute of Y if X does not have independent existence in the application.
(2) X and Y are related via an inheritance, aggregation, or association relationship if both X and Y have independent existence in the application.

Visualizing the Domain Model

- In this step, the classification result is visualized by using a class diagram. Once the classes, attributes. and relationships are identified and documented converting the result to a ctass diagram is an easy task.
- It shows the conversion rules and examples from the previous sections. Since the conversion is trivial, the drawing can be outsourced, or performed by an assistant to reduce cost.
- Convert the classification result obtained in into a class

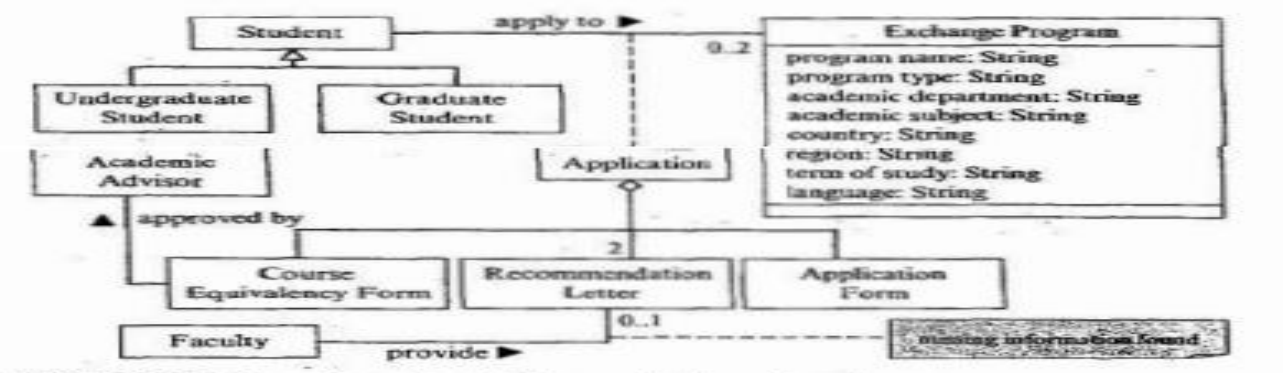


FIGURE 5.13 Converting classification result into a class diagram

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

http://stg-tud.github.io/eise/WS11-EiSE-07-Domain_Modeling.pdf
https://en.wikipedia.org/wiki/Domain_model
https://www.youtube.com/watch?v=M1e2XwSADDE
https://www.youtube.com/watch?v=DHJwT-sI0f4

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

BooksDavid Kung Object-Oriented Software Engineering:An Agile Unified Methodology McGraw-Hill Education 2013 Page No:117-130

Course Faculty

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

LECTURE HANDOUTS

CSE

II/IV

L-

Course Name with Code :Object Oriented Software Engineering/16CSD04

Course Faculty :

Unit : IIDate of Lecture:

Topic of Lecture:ARCHITECTURAL DESIGN

Introduction : (Maximum 5 sentences)

The software architecture defines the structure of the software system in terms of the subsystems and their interrelationships.

As pointed out that the beginning of this chapter the software architecture is the primary artifact for conceptualizing. constructing, managing, and evolving the system under developers.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- Domain Modeling

Detailed content of the Lecture:

Conceptualization.

The architecture defines the overall structure of the system. Therefore, in subsequent development activities, the architecture helps the development team to think of the system in'terms of its overall structure. Consider, for example. the N-tier architecture. It depicts the system as consisting of N layers of components, with each higher layer requesting services from the next lower layer.

Construction.

The architecture facilitates the construction of the software system because it lets the learn members know how to organize the software artifacts produced during the development process. Consider again the N~tier architecture for an interactive system, It typically consists of the' following layers, listed from high to low.

The presentation layer.

This layeris responsible for presenting the graphical user interface and system response the.

The business objects layer:

This layer is responsible for processing the business transactions represented by the use cases. '

The persistence storage layer.

This layer consists of objects that provide database-related functions such as object storage and retrieval.

The network communication layer.

This layer provides network communication-related functions. The responsibilities of and the dependencies.

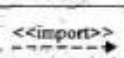
Managing:

The software architecture provides an architectural view for organizing the software artifacts

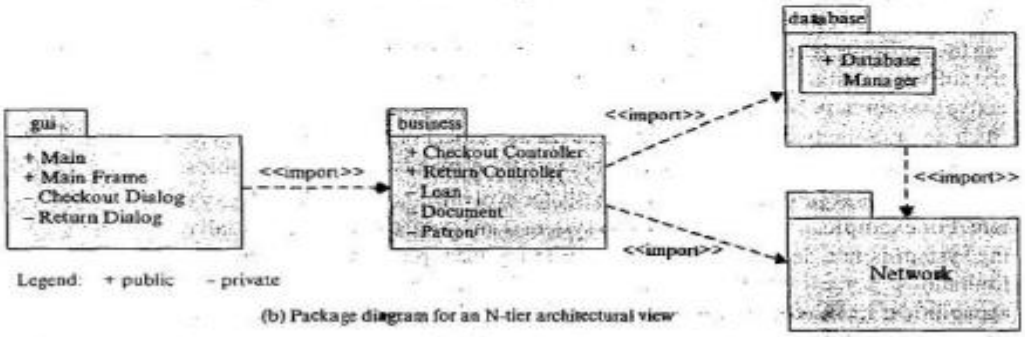
produced during the development process.

Evolving:

The architecture provides a basis for evolving and expanding the system. For example. a library information system is an interactive system. Initially, the system is not designed to support interlibrary loan, probably due to budget limitations.

Notion	Semantics	Notation	Relationships
Package	A logical grouping of software artifacts including classes, other packages, and other software artifacts of interest.		• A package may own software artifacts such as classes and other packages. • A package may import other packages.
import	A stereotyped relationship between two packages. The source package at the arrow tail imports the public elements of the destination package pointed to by the arrow head.		

(a) Package diagram notions and notations



(b) Package diagram for an N-tier architectural view

FIGURE 6.11 Package diagram for a library information system

- The package diagram in Library information system defines logic as organization or logical view of the classes of the library information system.
- Package diagrams help the team members understand which artifacts belong to which packages,
- The package hierarchy (wherein one package can own other packages) facilitates change control because the packages to be changed and the packages impacted can be identified at any level of the hierarchy.
- Without such a logical organization, configuration management of the classes and other artifacts would be more difficult.

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

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/
https://www.tutorialspoint.com/software_architecture_design/architecture_models.htm
https://www.youtube.com/playlist?list=PLAwxTw4SYaPkMTetlG7xKWaI5ZAZFX8fL

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

Books

David Kung Object-Oriented Software Engineering:An Agile Unified Methodology McGraw-Hill Education 2013 Page No:158-160

Journals

https://www.researchgate.net/publication/266139171
https://ieeexplore.ieee.org/document/1605177
http://web.mit.edu/richh/www/writings/hilliard99a-using-uml-for-AD.pdf

CourseFaculty

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

LECTURE HANDOUTS

CSE

II/IV

L-

Course Name with Code :Object Oriented SoftwareEngineering/16CSD04

Course Faculty :

Unit : IIDate of Lecture:

Topic of Lecture: Architectural DesignProcess Style

Introduction :

It showed that different types of systems require different software architectures. Therefore, it is important to select an architectural style that matches the system under development.

An architectural style is a generic architectural design that can be adopted or adapted for a system.

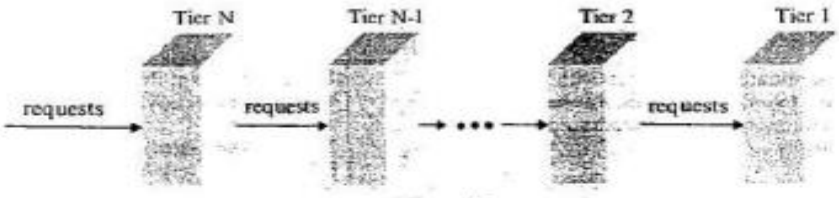
Prerequisite knowledge for Complete understanding and learning of Topic:

- Architectural Design

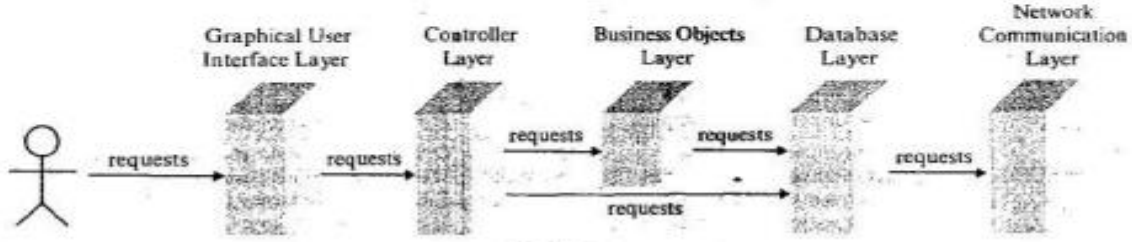
Detailed content of the Lecture:

N-tier architecture

This architectural style arranges the system components into a number of relatively independent, loosely coupled layers. Each layer has a well-defined functionality. It reduces change impact to other layers. It is useful for the design of interactive systems.



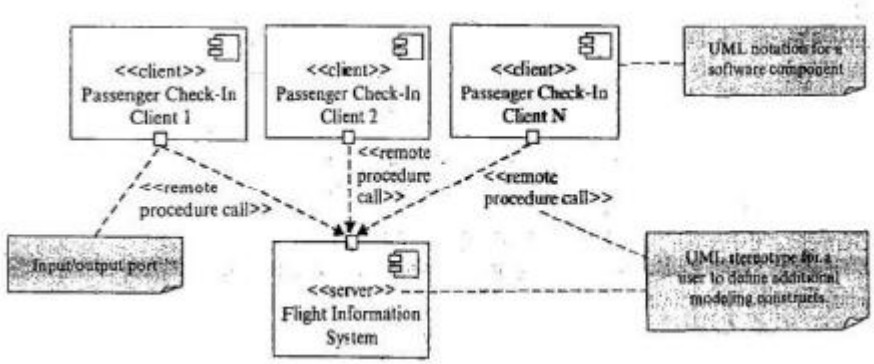
(a) N-tier architectural style



(b) An example N-tier architecture

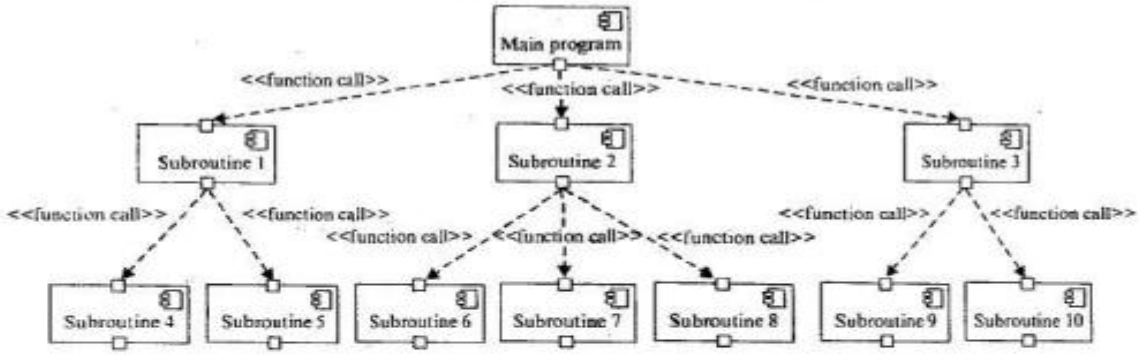
Client-server architecture

This architectural style consists of one server that provides services to a number of clients. The-clients know the server, the server does not know the clients, and the clients do not know each other. In this sense, it reduces the coupling of the clients and the server.

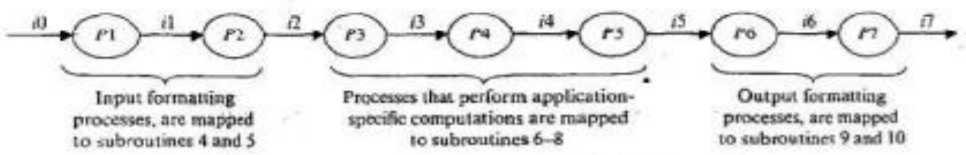


Main program and subroutines architecture

This architectural Style organizes the components of the system into a tree structure, in which computation begins with the root or main program and is carried out by the descendant's recursively down the tree, It is useful for designing transformational or work flow-oriented systems.



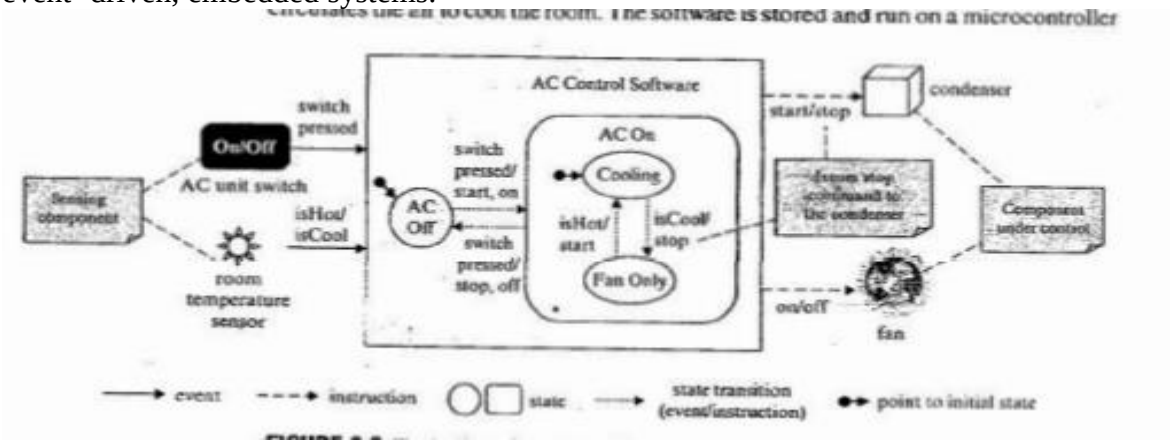
(a) A main program and subroutines architecture



(b) A data flow model and its mapping to the architecture design

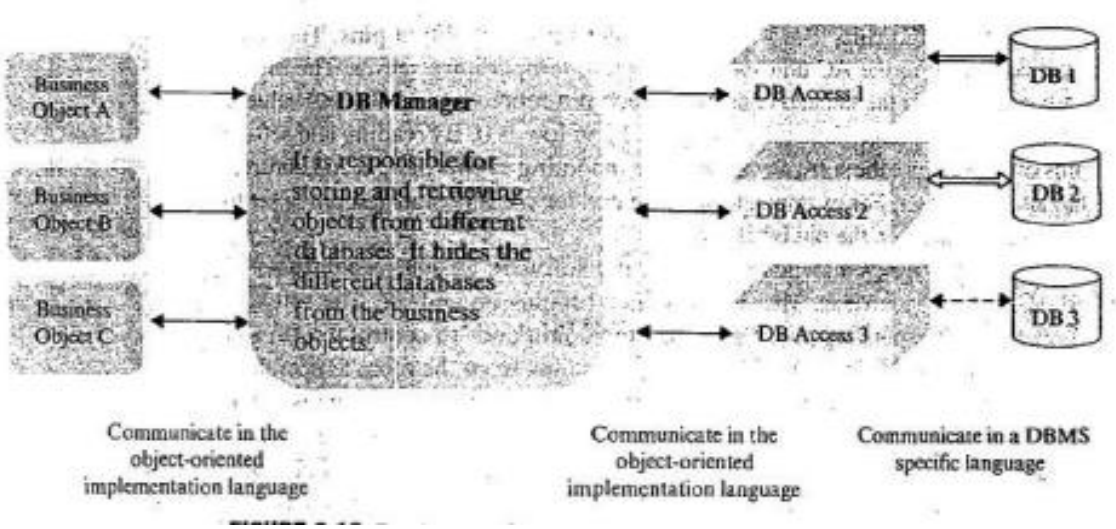
Event driven system architecture

This architectural style consists of a state- . .based controller that interacts with, and controls a number of components. The controller knows the components and vice versa, but tile components do not know each other. Idle action between the components is mediated by the controller. It is useful for designing event -driven, embedded systems.



Persistence framework architecture

This architectural style hides the databases and file systems by decoupling them from the objects that use them. That is, the objects are unaware of the existence of such storage devices; and hence, all changes to the databases and file system have no impact to the objects.



Other Architectural Styles

- Peer to Peer
- Pipe and Filter
- Blackboard
- Service Oriented Architecture
- Cloud computing Architecture

Specify Subsystem functions and interfaces

The interfaces between the subsystems are specified in this step, The specification of the interfaces defines the input and output of each subsystem including the number, types, and order of the input parameters arid similarly for the output.

Review the Architectural Design

The architectural design is reviewed to ensure that the design objectives and software requirements are satisfied.

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

https://www.geeksforgeeks.org/software-engineering-architectural-design/
https://cs.ccsu.edu/~stan/classes/CS410/Notes16/06-ArchitecturalDesign.html
https://www.youtube.com/watch?v=TzYYG06x9e0
https://www.youtube.com/watch?v=JLbo9Lvvy5M

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

Books:

4. David Kung Object-Oriented Software Engineering:An Agile Unified Methodology McGraw-Hill Education 2013 Page No:139-169

Journals

https://www.researchgate.net/publication/321675591
https://www.sciencedirect.com/science/article/pii/S187705091503183X

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code **:Object Oriented Software Engineering/16CSD04**

Course Faculty **:**

Unit **: IIDate of Lecture:**

Topic of Lecture:ARCHITECTURAL DESIGN PACKAGE DIAGRAM

Introduction :

The software architecture of a system or subsystem refers to the style of design of the structure of the system including the interfacing and interaction among its major components.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Domain Modeling

Detailed content of the Lecture:

IMPORTANCE OF ARCHITECTURAL DESIGN

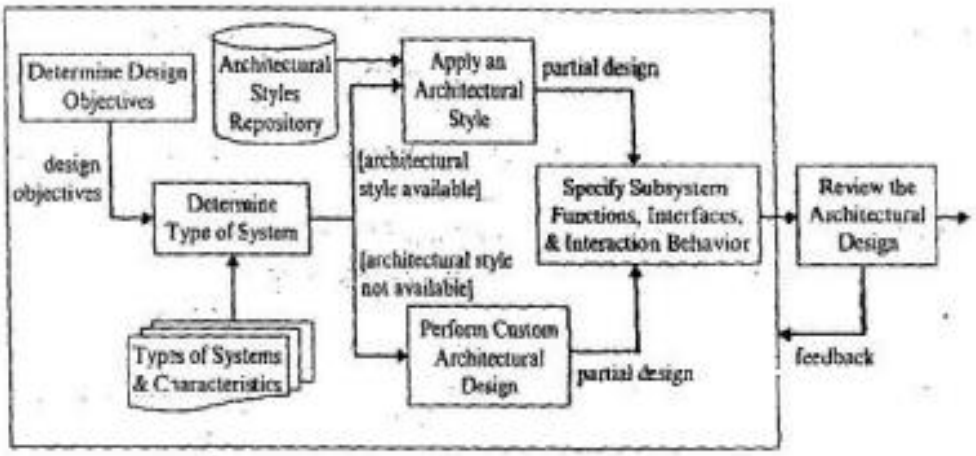
The importance of architectural design can never be overstated, as explained by the following story that took place many years ago.

ARCHITECTURAL DESIGN PROCESS

The architectural design process for a software system or subsystem is a decision-making, cognitive process.

It needs to consider many factors. The type of the system to be developed is an important consideration. Experiences show that the type of system influences the selection of the architectural style.

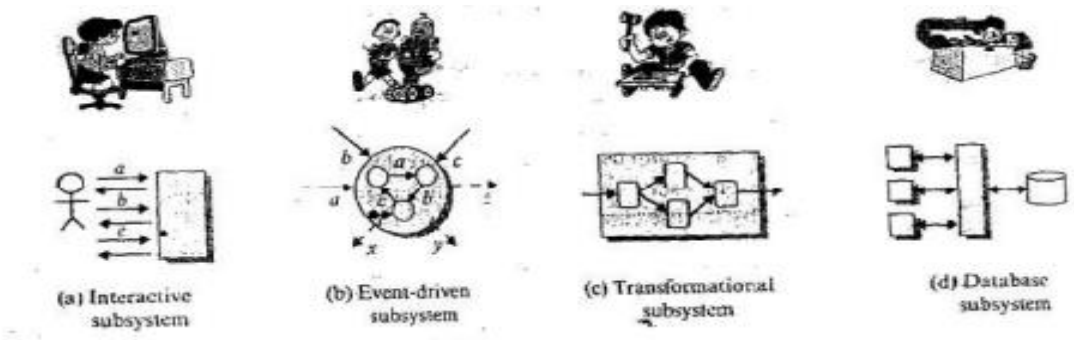
- Ease of change and maintenance
- Use of commercial off the shelf parts
- System Performance
- Security
- Reliability
- Software Fault Tolerance
- Recovery



- Determine design objective
- Determine type of System
- Apply an Architectural Design
- Specify Subsystems, interface and interaction behavior.
- Review the architectural design

Determine System Type:

The type of a system significantly influences the modeling. analysis design implementation and testing of a system.



Interactive Systems

The interaction between the system and the actor to carry out a business process consists of a relatively fixed sequence of actor requests and system responses as 'illustrated in Figure 6.2(a).

The system has to process and respond to each request from the actor.

Often, the system interacts with only one actor during the process of a use case.

Event-Driven Systems

The system receives events from and controls the external entities.

In general, event-driven systems do not have a fixed sequence of incoming requests; the requests arrive at the system randomly.

Transformational Systems

Transformational systems-can be. conceptually viewed as consisting of a network of information-processing activities, each of which transforms its input into its output as illustrated .

The network of activities may involve control flows that exhibit sequencing, conditional branching, and parallel threads as well as synchronous and asynchronous behavior.

Object-Persistence Subsystems

- It hides the database from the rest ofthe system and shields the rest of the system from changes to database implementation.

- Unlike the other three types of subsystems, a database subsystem is responsible only for storing and retrieving objects from the database. It does little or no business processing except in a few cases when doing so can substantially improve performance, such as when a large number of records needs to be updated
- A database subsystem is capable of efficient storage, retrieval, and updating of a huge amount of structured and complex data.

Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/software-engineering-architectural-design/
https://www.tutorialspoint.com/software_architecture_design/introduction.htm
https://www.youtube.com/watch?v=kerqiiJZcm0

Important Books/Journals for further learning including the page nos.:
Books:
David Kung Object-Oriented Software Engineering:An Agile Unified Methodology McGraw-Hill Education 2013 Page No: 139-169
Journals
https://www.journals.elsevier.com/journal-of-systems-architecture
https://ijcsmc.com/docs/papers/March2016/V5I3201613.pdf
https://www.researchgate.net/publication

Course Faculty

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

LECTURE HANDOUTS

CSE	II/IV
	L-

Course Name with Code :**Object Oriented Software Engineering 16CSD04**
CourseFaculty :

Unit : **IIDate of Lecture:**

Topic of Lecture:APPLYING SOFTWARE DESIGN PRINCIPLES

Introduction :

Software design principles are widely accepted guiding rules for software design – correctly applying these principles can significantly improve software quality.

Prerequisite knowledge for Complete understanding and learning of Topic:

- Architectural Design
- Software design

Detailed content of the Lecture:

Software design principles are collective wisdom acquired and validated by the software engineering community during decades of software research and development (R&D). They are valuable assets of the community.

The next several sections are devoted to the study of software design principles including design for change, separation of concerns, information hiding, high cohesion, low coupling, and keep it simple and stupid (KISS)..

Design for Change

The design for change principle is rooted on the fact that change is the way of1ife. Numerous events could cause changes to a system. A few of these are given below-to motivate:

- Changes to software requirements are needed to respond to changes in the business environment.
- Changes to the software system are needed to fix problems in the system.
- Changes to the system are needed due to changes in hardware, platform, system operating environment, and the like.

Separation of Concerns

Separation of concerns was proposed by Edsger Dijkstra as a problem-solving principle, that is; focusing on one aspect of the problem in isolation rather than talking all aspects simultaneously.

- first(): This function .sets the cursor to r~fer to the first element of the aggregate.
- next(): This function advances the cursor to the next available element.
- isDone(): boolean This function returns true if all elements of the aggregate are visited.
- getElement(): Object This function returns the element referred to by the cursor.

High Cohesion

- The high cohesion principle came from modular design in the conventional structured analysis and structured design paradigm.
- In structured design, the software system is-decomposed into a treelike hierarchy of modules in which higher-level modules call-lower-level modules and synthesize-the results retuned from the lower-level modules.

Low Coupling

- The Low Coupling Principle also came from the structured analysis and structured design paradigm.
- In structured design, coupling measures the degree of run-time effect due to dependencies and interaction between the modules, in other words, the degree of impact of the run-time behavior of a given module on the run-time behavior of other modules.

GUIDELINES FOR ARCHITECTURAL DESIGN

- Adapt an architectural style when possible
- Apply software design principles.
- Apply design patterns.
- Check against design objectives and design principles.
- Iterate the steps if needed.

ARCHITECTURAL DESIGN AND DESIGN PATTERNS

- Design patterns are proven design solutions to commonly encountered design problems. As such, design patterns are widely used in architectural design and architectural styles.
- The persistence framework combines several design 'patterns to accomplish a number of design objectives such as design for change low coupling, high cohesion, separation of concerns, and designing stupid objects.
- The patterns used include bridge, command, proxy, and template method. Part (Applying Situation-Specific Patterns) presents these patterns as well as the design of the persistence framework.

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

http://user.it.uu.se/~carle/softcraft/notes/SoftwareDesignPrinciples.pdf
http://ecomputernotes.com/software-engineering/principles-of-software-design-and-concepts
https://www.youtube.com/watch?v=WV2Ed1QTst8
https://www.youtube.com/watch?v=HLFbeC78YlU

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

Books:

David Kung Object-Oriented Software Engineering:An Agile Unified Methodology McGraw-Hill Education 2013 Page No:160-166

Journals

https://www.researchgate.net/journal/1945-3116_Journal_of_Software_Engineering_and_Applications
https://www.ijser.org/researchpaper/Effect-of-SOLID-Design-Principles-on-Quality-of-Software-An-Empirical-Assessment.pdf

CourseFaculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : Object Oriented Software Engineering-16CSD04

Course Faculty :

Unit : III **Date of Lecture:**

Topic of Lecture : Deriving Use Cases from Requirements

Introduction : (Maximum 5 sentences) :

- The fundamental goal of each software project is to build and deliver the right product for target users.
- But : What is a 'right product'?
- The right product is a product that the customers want, need, and desire. Unfortunately, no one knows at upfront what they want and need, including the customers themselves.
- A systematic approach that helps you identifies customers needs.
- It involves an upfront recognition of business goals to be satisfied, and gradually a discovery of requirements based around the goals.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Agile Principles
- Architectural Design
- Design Patterns
- UML

Detailed content of the Lecture:

Use case and use case diagram

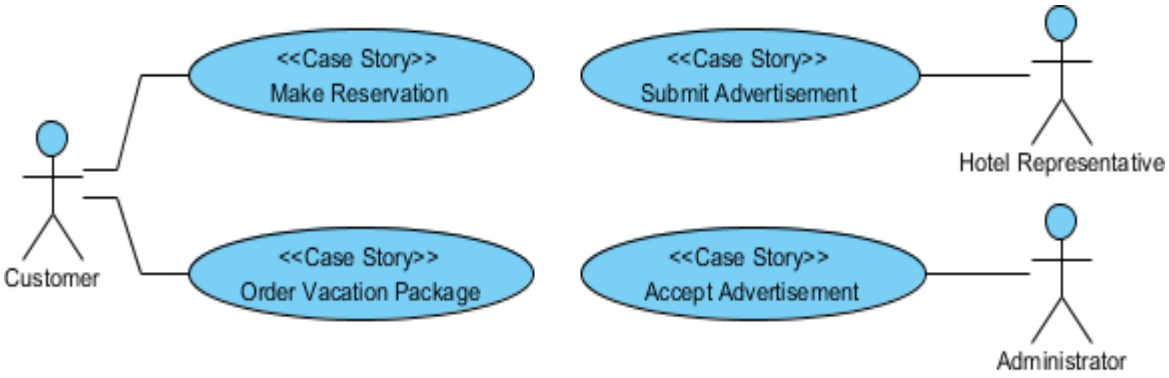
What is use case?

- A use case describes a specific business goal to be satisfied by the system to be built. Graphically, it is an oval with a name, which looks simple but is yet the most commonly used tool in managing business goals or project goals.

What is use case diagram?

- A use case diagram is a kind of Unified Modeling Language (UML) diagram created for requirement elicitation.

- Use case diagram provides a graphical overview of goals (modeled by use cases) users (represented by actors) want to achieve by using the system.
- Use cases in a use case diagram can be organized and arranged according to their relevance, level of abstraction and impacts to users.
- They can be connected to show their dependency, inclusion and extension relationships.
- The main purpose of modeling use case with use case diagram is to establish a solid foundation of the system by identifying what the users want.
- Based on the result, you can move forward to study how to fulfill those user needs.



What is user story?

- Anyone who has experience in software development would probably have suffered from communication issues with stakeholders.
- User story is a great way of opening discussion with stakeholders for ensuring the development team knows what stakeholders want.
- User stories created by the product owner capture "who", "what" and "why" of a requirement simply and concisely, which is typically written in natural language in a non-technical format.
- Agile development has entered into the mainstream of development approach hand-in-hand with user stories for requirement discovery.

Discovering user stories with use cases

- It is important to note that use cases alone represent goals but not the actual requirements to be supported. Nevertheless, use cases provide a great starting point to the discovery of requirements.

Here are the benefits:

- Use cases provide a clear project scope. The chance of identifying requirements beyond the project scope can be reduced
- Requirements derived from use cases are guaranteed to be aligned with the business vision and goals.
- Traceability between use case and requirements helps clarify the rationale of requirements at any moment of software project.
- **To summarize:** Use cases can be effective when you use it as a tool for requirements discovery and management.

Drawing Use Case Diagram in Visual Paradigm

- You can develop a use case model and write user stories with Visual Paradigm. We will make use of a hotel reservation system as an example.
- Let's start by drawing a use case diagram.
- Create a new project in Visual Paradigm by selecting Project > New from the toolbar. In the New Project window, name the project Hotel Reservation System and click Create Blank Project at the bottom.
- To create a Use Case Diagram, select **Diagram > New** from the toolbar. In the **New Diagram** window, select **Use Case Diagram** and click **Next**.
- Keep **"Blank"** selected and click **Next**. Enter System Use Cases as diagram name and click **OK**.
- Press on **Actor** in the diagram toolbar. Drag it onto the diagram to create an actor and name it Customer.
- The system will let users make a reservation, which is a use case of the system. Let's create a use case for it. Move the mouse pointer over the Customer actor.
- Press on the **Resource Catalog** icon at the top right and drag it out.
- Select **Association -> Use Case** in Resource Catalog.
- Release the mouse button to create the use case. Name it Make Reservation. The association between actor and use case indicates that the actor will interact with the system to achieve the use case associated.

Video Content / Details of website for further learning (if any):
https://www.visual-paradigm.com/tutorials/writingeffectiveusecase.jsp
https://www.youtube.com/watch?v=HshfGCgWaE4

Important Books/Journals for further learning including the page nos.:
David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 Page no: 172-198

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : Object Oriented Software Engineering-16CSD04

Course Faculty :

Unit : III **Date of Lecture:**

Topic of Lecture: Actor-system interaction modeling

Introduction : (Maximum 5 sentences) :

- Actor-system interaction modeling is modeling and design of how the system interacts with the actors to carry out the use cases.
- Actor-system interaction modeling is accomplished by constructing a two-column table that describes, for each interaction, the actor input and actor action, and the system response.
- Focuses on the modeling and design of such interaction behavior.
- Derive the use cases from the requirements and how to specify the scope of each use case. The results are referred to as abstract use cases and high-level use cases.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Use cases and Use cases Diagram
- Actor
- Action
- Operation

Detailed content of the Lecture:

USE CASES ARE MODELED WITH THREE LEVELS OF ABSTRACTION:

1) Abstract use case: using a verb and a noun phrase

2) High level use case: stating exactly when and where the use case begins and when it ends using TUCBW ... (This use case begins with ...) and TUCEW ... (This use case ends with ...)

3) Expanded use case: describing step by step how the actor and the system interact to accomplish the business task using a two column table.

ACTOR-SYSTEM INTERACTION MODELING :

Actor-system interaction modeling is the modeling and design of how the system interacts with-the actors to carry out the use cases.

As shown in Figure, the left column specifies the actor input and/or actor actions; the right column specifies the corresponding system responses. More specifically, the two-column tabular specification

of an expanded use case illustrates the following :

1. **Use-case ID and name.** This is shown at the top of the table in Figure.

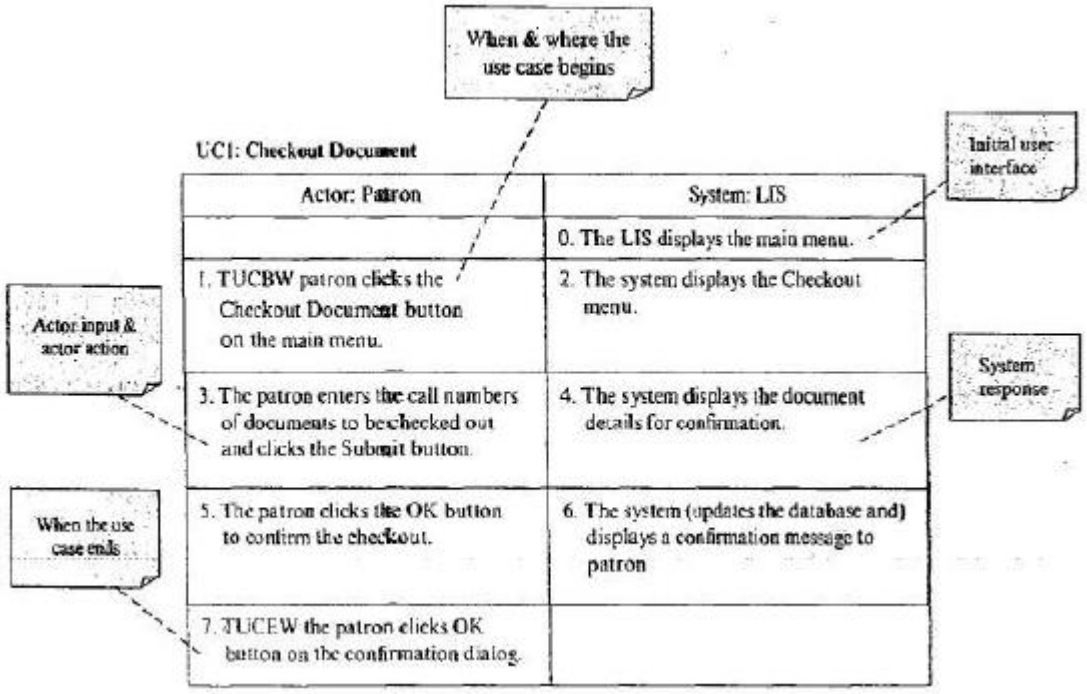
2. **The initial state of the user interface.** Step 0 on the right column specifies. The initial state of the user interface before the use case begins. It is important to specify the initial state of the user interface because: (a)' it tells the developer what the system should display to-the actor; and (b) it tells the actor what he or she will see before the use case begins.

3. **When and where to start the use case.** Step I on the left column specifies when and where the use case begins. This is the TUCBW clause of the high-level use case.

4. The actor input and actor action at each step of the interaction. This is specified in the left column on each row of the two-column table.

5. The corresponding system response. This is specified in the right column on the corresponding row of the two-column table.

6. When the use case ends. The last entry of the left column specifies when the use case ends. This is the TUCEW clause of the high-level use case.



IMPORTANCE OF ACTOR-SYSTEM INTERACTION MODELING

The usefulness of the expanded use case specification includes the following:

1. It specifies the actor-system interaction or system's interactive behavior that the subsequent design, implementation, and testing can follow.

2. It can be used to generate the preliminary user's manual. This is because the expanded use case describes exactly how the user will use the system to accomplish a business task..The preliminary user's manual facilitates a potential user to experiment with a prototype of the system.

3. If updated timely to reflect changes in actor-system interaction during the subsequent design and implementation phases, the updated expanded use case specification can be used to generate the as-built user's manual. This reduces the increment or system deployment effort, cost and time.

4. It can be used to generate use case-based test cases.

STEPS FOR ACTOR-SYSTEM INTERACTION MODELING

The main activity of actor-system interaction modeling is constructing expanded use cases for the use cases allocated to the current iteration. It involves the following steps:

• **Step 1.**Initialize a two-column table for the expanded use case being constructed.

• **Step 2.** Specify each of the actor-system interaction steps until the system produce, the response specified in the TUCEW clause.

• **Step 3.** Review the actor-system interaction using a review checklist.

Initializing a Two-Column Table

- Draw a two-column table and show the use case ID and use case name at the top of the table.
- Name the headers of the left and right columns with the role name of the actor and the system/subsystem name, respectively.
- Enter the TUCBW and TUCEW clauses of the corresponding high -level use case to the second and third entries of the left column, and label them step 1 and step 3 respectively.
- The step number 3 will increase as more steps are inserted. Leave the first entry of the left column, that is, the entry right beneath the left column header, blank.
- Next, infer the initial system display according to the TUCBW clause of the use case and specify this in the first entry of the right column.
- Label this step as step 0.
- Figure show the result of this step for a library information system (LIS). Note that main menu is referred to in the TUCBW clause in step 1, therefore, step 0 is 0.
- The LIS displays the main menu.

UC1: Checkout Document

Actor: Patron	System: LIS
	0. The LIS displays the main menu.
1. TUCBW patron clicks the Checkout Document button on the main menu	2.
3. TUCEW patron clicks the OK button on the confirmation dialog.	

Specifying Actor-System Interaction Steps

- In this step, the actor-system interaction steps arc specified.
- It begins with the TUCBW clause in step 1.
- The corresponding system response is derived and entered as step 2 in the right column.
- The result is written as "the system displays .... ,.
- In general, the system displays the result or a dialog to acquire actor input.
- If actor input is required, then a row is inserted and the actor input and actor action are specified.
- This process is repeated for the remaining steps until the system produces the response specified

in the TUCEW clause.

- Sometimes, the system requires the actor to enter information about a domain concept.
- Such information is usually found in the domain model.

Reviewing Actor-System Interaction Specifications :

The expanded use cases produced in the current iteration are reviewed using the following review checklist:

1. Are there a use case ID and a use case name for the specification? Do they match with the use case ID and name in the requirement-use case traceability matrix and the corresponding high-level use case?

2, Are the actor and system correctly identified and specified and match with the counterparts in the high-level use case?

3. Does the expanded use case specify the initial system display in step O?

4. Are the TUCBW and TUCEW clauses matched with their counterparts in the high-level use case?

5. Does the use case begin and end with the actor on the left column?

6. Are there blank entries during the course of interaction between the actor and the system?

7. Does the expanded use case correctly and adequately specify the actor-system interaction to carry out the business process?

8. Do the left-column steps clearly and correctly specify the actor input, and actor actions such as clicking the OK button?

9. Do the right -column steps clearly and correctly specify the system responses to the actor?

Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/layers-of-osi-model/
https://www.youtube.com/watch?v=EzLMMsRR6Js

Important Books/Journals for further learning including the page nos.:
David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 PAGE NO:200-213

Course Faculty

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-

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : Object Oriented Software Engineering-16CSD04

Course Faculty :

Unit : III **Date of Lecture:**

Topic of Lecture: Object Interaction Modeling

Introduction : (Maximum 5 sentences) :

- Object Interaction Modeling demonstrates the dynamic behavior that occurs between objects by integrating the static Class Model with use cases.
- The Class Model defines the internal structure of objects but says nothing about how they inter-operate whereas use cases depict the operations between objects in the problem domain without concern for the internal composition of the objects themselves.
- You can model different aspects of the system domain, reflecting system needs from different user perspectives by depicting the interaction and messaging between objects in the system.
- Collectively these views and their underlying definitions are referred to as the Object Interaction Model.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- Actor-system interaction modeling
- Object
- Class
- Sequence diagram

Detailed content of the Lecture:

- The integration of use cases and Class Diagrams in the development of Sequence Diagrams is an iterative process during which the Use Case Model (the users view) and the Class Model (the developers view) is cross checked with user requirements and refined.
- Object Interaction Modeling in Modeler involves two types of modeling diagram derived from the Class Model and use cases:
- Sequence Diagram used to describe a use case or an operation in terms of the constructs of sequence, selection and iteration; the passage of time is depicted by an invisible time axis running downward through the diagram
- Communication Diagram used to describe a scenario or path within a use case; comprising objects and message flows between objects, in a snapshot of a time-lapse interaction; each path through the structured language of a Sequence Diagram can be modeled by its own

Communication Diagram; typically only the most important scenarios are modeled with Communication Diagrams.

- A Sequence Diagram will map directly to one use case. A typical use case will consist of a set of scenarios, or paths, through the system being modeled representing different options within the use case. Each execution path can be represented by a unique Communication Diagram. Therefore several Communication Diagrams can map to one Sequence Diagram.

OBJECT INTERACTION MODEL

- Object Interaction Modeling demonstrates the dynamic behavior that occurs between objects by integrating the static Class Model with Use Cases.
- For information about Object Interaction Modeling in Modeler, see Object interaction modeling.
- You can view the Object Interaction Model part of a model through the Object Interaction Model folder in the Relationships pane.

STEPS FOR OBJECT INTERACTION MODELING

- The steps for OIM are depicted in Figure and outlined in the following list. They are performed for the use cases that are allocated to the current iteration.

Step I. Collecting information about the existing business processes.

- In this step, the development team collects and studies information about the existing business processes of the use cases.

Step 2. Specify scenarios for the nontrivial steps of the expanded use cases

- In this step, nontrivial steps of the expanded use cases are identified. Scenarios for such steps are specified. A scenario is a series of declarative sentences that describes how the objects interact with each other to carry out a non trivial step.
- The input of this step is the expanded use cases for the current iteration and the information collected in the last step. The output of this step is a list of scenario descriptions.

Step 3. Constructing scenario tables

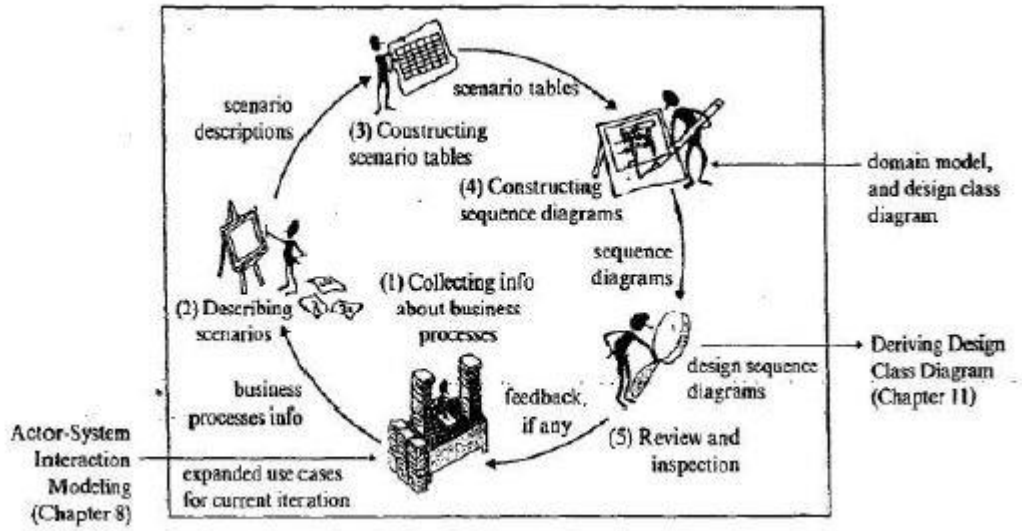
- In this step, a tabular representation for each scenario, called a scenario table, is produced as an aid to sequence diagram construction.
- The input of this step is the list of scenario descriptions. The output of this step is a set of scenario tables.

Step 4. Deriving sequence diagrams from scenario tables.

- In this step, the scenario descriptions or scenario tables are converted into UML objects are determined.
- The input of this step is the scenario descriptions or scenario tables, the design class diagram produced in previous iterations, and the domain model.
- The output of this step is a set of sequence diagrams.

Step 5. Reviewing the object interaction model.

- In this step, the object interaction models are reviewed and revised for consistency, completeness, and correctness.
- The sequence diagrams are then used to derive the classes to be implemented.
- Also derived are attributes and operations of the classes, and dependency relationships between the classes. The results are depicted in a UML class diagram, called the design class diagram (DCD). The DCD serves as the design blueprint for implementation and testing.



Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/layers-of-osi-model/
https://www.youtube.com/watch?v=EzLMMsRR6Js

Important Books/Journals for further learning including the page nos.:
David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 PAGE NO:216-246

Course Faculty

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

LECTURE HANDOUTS

L

CSE

II/IV

Course Name with Code : Object Oriented Software Engineering-16CSD04

Course Faculty :

Unit : III **Date of Lecture:**

Topic of Lecture: Applying Responsibility

Introduction : (Maximum 5 sentences) :

- Design patterns are abstractions of proven design solutions to commonly encountered design problems.
- The controller, expert, and creator patterns are applicable to almost all object oriented systems.
- It will discuss relevant software design principles to help understand what is considered a good design.
- In particular, it will discuss problems associated with some of the commonly seen design sequence diagrams.
- To solve these problems, several design patterns are introduced and how to apply these patterns to improve the design is illustrated.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Object Interaction Modeling
- UML
- Use Case
- Agile Principles

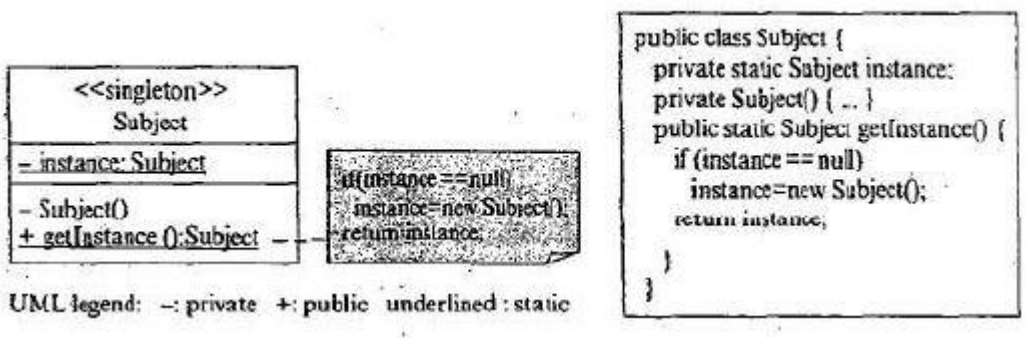
Detailed content of the Lecture:

- Object interaction modeling (OIM), a basic methodology because it does not take into account the various software design principles.
- This is done on purpose to make the steps easy to understand and easy to follow.

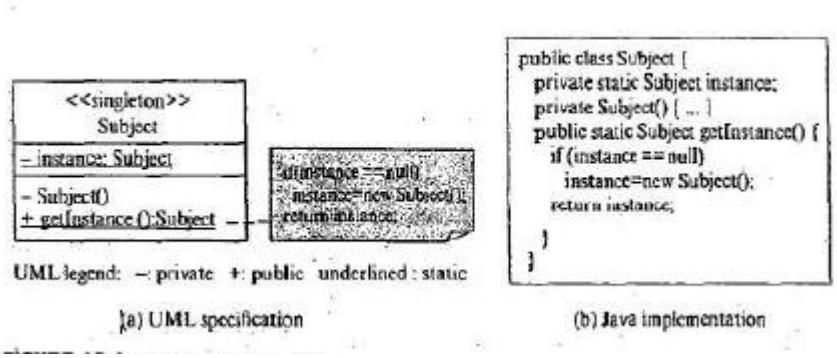
WHAT ARE DESIGN PATTERNS?

- Human beings have used patterns for: II long time. For example, farmers have used cloud patterns to predict weather, and stock investors use chart patterns to predict price movements of stocks and mutual funds.

- The architectural design patterns were first studied by Christopher Alexandra, who discovered that buildings that can withstand natural disasters have something in common.
- They all exhibit a set of design Ideas, which he formulated as design patterns.
- Design patterns are abstractions of proven design solutions to commonly encountered design problems.
- First, a pattern is a design abstraction, as opposed to a concrete design.
- This enables a pattern to solve many similar design problems. Each solution is an instance of the pattern.
- Each pattern is also a design solution; it solves a design problem or a class of similar design problems.
- Tie design problem is unique to the pattern.
- A pattern is also proven design solution-, that is, its effectiveness is established by practical applications, not claimed.
- Finally, a. pattern solves a commonly encountered design problem and hence, it can be applied again and again to solve many similar design problems 10 a Wide variety of applications. .
- The software engineering community recognized the idea and began tile R&D in software design patterns in the I980s.
- To date, the most well-known and influential collection of software design patterns remains the 23 so-called Gang of Four (GoF) patterns.
- These patterns are called Gang of Four patterns because the book collecting them has four authors.
- Each pattern has a name, comprised of an abstraction of the design problem and the design solution, for example, the singleton pattern solves the following design problem: "How does one design a class that has at most one globally accessible instance.
- This design problem is common in many practical applications.
- For example, a system needs only one system configuration object and many components need configuration information.
- Therefore, it is desirable to make it globally' accessible system log file and the catalog of a library are other applications of singleton.
- A UML class diagram that describes the pattern and Figure shows an Implementation in Java.



- The Subject class is the application class that should have at most one globally accessible instance, for example, the System Configuration, the System Log or the Library Catalog class.
- It has a private instance of its own type.
- This instance is initially null.
- The Subject class has a private constructor.
- This ensures that ether objects cannot create an instance of the Subject class.
- The getInstance () function ensures that at most one instance is created. It allows other objects access to the single instance globally through static calls.
- As Illustrated above, patterns are often described using class diagrams and sometimes also sequence diagrams.
- The class diagram specifies the participants, their roles and responsibilities, and how they relate to each other.
- The sequence diagram describes how the participants interact with each other to solve the design problem. UML notes are commonly used to provide additional information.



(a) UML specification (b) Java implementation

WHY DESIGN PATTERNS?

- Patterns are proven design solutions to commonly encountered design problems.
- In other words, patterns are reusable software elements. Patterns can be combined to solve large complex design problems.
- In addition, patterns offer a number of benefits.
- First, patterns improve team member communication because the pattern names effectively convey the design problems and solutions. Improved communication leads to improvement in teamwork and elevates team moral.
- Many patterns implement software design principles. Therefore, patterns improve the structure and behavior of software systems.
- Components designed and implemented with patterns are easy to understand, test, and maintain. Patterns make reusable designs.
- Successful reuse of well-designed and well-tested software components improves software productivity and software quality.

- Many experiences indicate that the use of design patterns significantly enhances the development team's ability to tackle complex design problems.
- Patterns empower less-experienced developers because they can apply patterns to produce high-quality software. In summary, patterns improve software productivity and software quality, and reduce software cost and time to market.

Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/layers-of-osi-model/
https://www.youtube.com/watch?v=EzLMMsRR6Js

Important Books/Journals for further learning including the page nos.:
David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 PAGE NO:251-273

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : Object Oriented Software Engineering-16CSD04

Course Faculty :

Unit : III **Date of Lecture:**

Topic of Lecture: Assignment patterns : Specification

Introduction : (Maximum 5 sentences) :

- Patterns are proven design solutions to commonly encountered design problems.
- In other words, patterns are reusable software elements.
- Patterns can be combined to solve large complex design problems. In addition, patterns offer a number of benefits.
- Design patterns are abstractions of proven design solutions to commonly encountered design problems.
- The controller, expert, and creator patterns are applicable to almost all object oriented systems.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Design patterns
- UML Diagram
- Creator Pattern
- The Domain model

Detailed content of the Lecture:

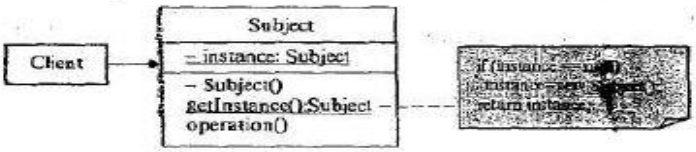
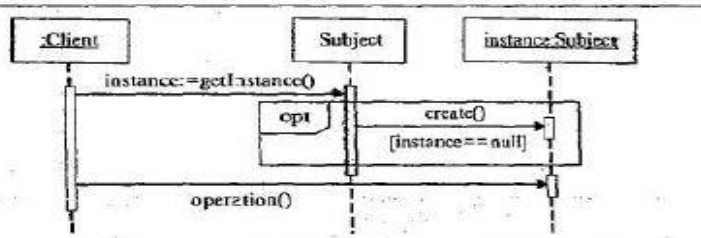
- The Gang of Four (GoF) patterns are situation-specific patterns, meaning that each pattern solves a specific class of design problem.
- For example, there are many applications of the singleton pattern.
- In addition, two or more patterns can be combined in an infinite number of ways to solve countless design problems.
- The GoF patterns are classified into three categories-that is creational patterns, structural patterns, and behavioral patterns, Creational patterns are useful for constructing complex, or special objects.
- For example, how to construct or initialize a complex class structure that involves different

classes and complex relationships between the classes, Structural patterns are useful for solving structural design problems.

- An example is how to represent a complex class structure such as a block diagram. Behavioral patterns are used to solve behavioral or algorithmic design problems.
- Examples are how to schedule the execution of operations, how to process events, and how to perform analysis algorithms on elements of a complex class structure.
- Another set of well-known software design patterns is the General Responsibility- Assignment Software Patterns (GRASP), published by Craig Lannan.
- Unlike GoF patterns, GRASP patterns are general responsibility-assignment patterns.
- General, because they can be applied to design almost every software system.
- Responsibility-assignment, because they address an important object-oriented design problem.
- That is, which object should be assigned a given responsibility so that the resulting design will exhibit properties advocated by software design principles.
- GRASP patterns are the controller, expert, and creator patterns.

PATIERN SPECIFICATION

- A pattern specification is a structured description of a Pattern to facilitate understanding and application of the pattern.
- For example, below Figure shows the specification of the singleton pattern.
- A pattern specification describes the important or useful aspects of a pattern.

Name	**Singleton**
Type	**GoF/Creational**
Specification	
Problem	**How does one design a class that has only a limited number of globally accessible instances?**
Solution	**Make the constructor of the class private and define a public static method to control the creation of the i**
Design	
Structural	
Behavioral	
Roles and Responsibilities	**• Subject: It provides a public static getInstance() method for the client to retrieve its instance. • Client: It calls the getInstance() method of Subject to retrieve the instance and calls its operation().**
Benefits	**• It limits the number of instances. • It supports global access to the instances.**
Liabilities	**• Concurrent update to the shared instance may cause unwanted effect.**
Guidelines	
Related Patterns	**• Singleton limits the number of instances of a class. Flyweight supports numerous occurrences of an object. Prototype reduces the number of classes. • Visitor is often a singleton.**
Uses	**Singleton is used in many applications. Calendar, Desktop, Runtime and Toolkit are [illegible] APIs that apply the singleton pattern.**

Name and type:

- These specify the name and the type of the pattern. The pattern type indicates the family of the pattern. For example, GoF, GRASP, or other type of patterns
- GoF patterns are further divided into creational, structural, and behavioral patterns.

Specification :

- This section specifies the design problem and design solution of the pattern.

Design:

- This section describes the structural design and the behavioral design of the pattern. A UML class diagram is used to describe the structural design. The behavioral design is described by a sequence diagram or texts.
- In addition, the classes are described in terms of their roles and their responsibilities in solving the design problem.

Benefits and liabilities:

- These describe the advantages of applying the pattern, and any potential problems.

Guidelines:

- Sometimes useful information for applying the pattern is provided.

Related patterns :

- Patterns that are related in various ways are described here.

Uses :

- General or specific applications of the pattern may be described.

Video Content / Details of website for further learning (if any):
https://whatis.techtarget.com/definition/GRASP-General-Responsibility-Assignment-Software-Patterns
https://www.youtube.com/watch?v=ViT0o4JSR7c

Important Books/Journals for further learning including the page nos.:
David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 PAGE NO:252-254

Course Faculty

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-

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : Object Oriented Software Engineering-16CSD04

Course Faculty :

Unit : III **Date of Lecture:**

Topic of Lecture : Controller-Expert Pattern

Introduction : (Maximum 5 sentences) :

- The controller pattern is used with almost every use case.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- Creator
- Controller Pattern
- Creator Pattern
- The Domain model
- Bloated controller

Detailed content of the Lecture:

THE CONTROLLER PATTERN

- It addresses the problem of how to design software systems that allow their user interfaces and business objects to change independently without affecting one another.
- Another design problem addressed by the pattern is how to support multiple user interfaces such as a desktop user interface, a web-based interface, or others. The controller pattern is a special: case of the well-known model-view-controller (MVC) pattern. That is, it is an application of the MVC to the handling of actor requests of a use case.

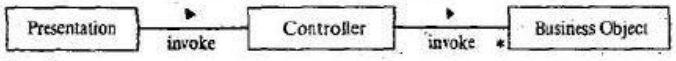
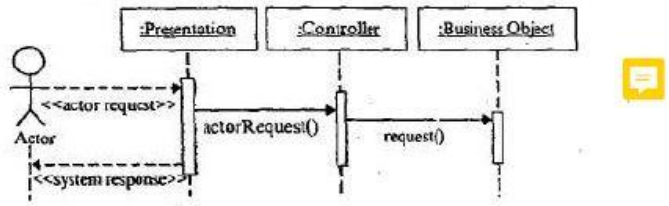
What Is a Controller?

- The Presentation and the business objects are now decoupled.
- As a consequence, changes to one will not affect the other. Supporting multiple types of presentation is easy.
- To add a new type of presentation, one need only to design and implement the new presentation and have it-deliver the actor request to the controller.
- This greatly facilitates software evolution or enhancement maintenance. The responsibility to

handle an actor request 'is removed from the presentation and assigned to the controller.

Applying the Controller Pattern

- To apply the controller pattern to the design in Figure IOJ, one simply introduces a controller in between the presentation and the business objects.
- In addition, the business logic is removed from the presentation and assigned to the controller. Notice that the Checkout GUI object now is only responsible for presenting information patron.
- The responsibility to process the checkout request is assigned to the Checkout Controller object, which interacts with the DBMgr and the other business objects to fulfill the responsibility.
- It is instructional to briefly discuss the design in Figure with respect to -the design principles presented.
- The discussion is aimed to illustrate how to evaluate a design using software design principles. In addition, the discussion helps the student understand the design principles.
- These are required attributes of a software architect.

Name	Controller
Type	GRASP
Specification	
Problem	Who should be responsible for handling an actor request?
Solution	Assign the responsibility to handle the request to a dedicated class called the controller.
Design	
Structural	
Behavioral	
Roles and Responsibilities	• Business Objects: Object classes responsible for the business logic of an application. • Controller: A class dedicated to handle designated actor requests. It takes requests from the presentation and works with the business objects to fulfill the request. A use case controller is dedicated to handle all actor requests of a given use case. It keeps track of use case state. • Presentation: A class responsible for interacting with an actor of the system. It delegates the actor requests to the controller and delivers the responses from the controller to the actor.
Benefits	• It decouples the presentation and the business objects. • It reduces the change impact of presentation and business objects to one and other. • It supports multiple presentations. • It keeps track of use case state.

Benefits	• It decouples the presentation and the business objects. • It reduces the change impact of presentation and business objects to one and other. • It supports multiple presentations. • It keeps track of use case state.
Liabilities	A controller may be assigned too many responsibilities, resulting in a so-called bloated controller. A bloated controller is complex, difficult to understand, implement, test and maintain.
Guidelines	• Adopt use case controllers whenever possible. • Avoid using one controller for more than one use cases. • The controller should collaborate with, and delegate responsibilities to business objects.
Related Patterns	• Controller is a special case of the Model-View-Controller or MVC pattern. • A controller can use State to keep track of use case state.
Uses	In the design of all interactive systems to decouple the presentation from business objects.

1. **Design for Change:**

- Changes to the business logic or business objects will have little impact to the presentation, provided that the interface and interaction behavior of the Checkout Controller are not changed.

2. Separation of Concerns:

- Separation of concerns is well supported by the design.
- The Checkout object now deals with only the presentation aspect while the Checkout Controller is responsible for processing the Checkout Document use case.
- In the previous design, both concerns are assigned to the Checkout GUL

3. High Cohesion:

- Previously the responsibilities of presenting information to the patron and processing the Checkout Document use case are assigned to the Checkouts GUL.
- But these two sets of functionality do not belong to a single core function. Therefore, the cohesion of the previous design is surely not functional cohesion.
- In the new design, both the CheckoutGUI and the Checkout Controller exhibit functional cohesion.

4. Designing "Stupid Objects."

- The new design exhibits the principle of keep it simple and stupid, especially, designing "stupid objects." Previously, the Checkout knew how to present information to the patron as well as how to process the business logic.
- In the improved design, each of the Checkout GUI and the Checkout Controller knows only one thing, presenting information and processing the Checkout Document use case, respectively.

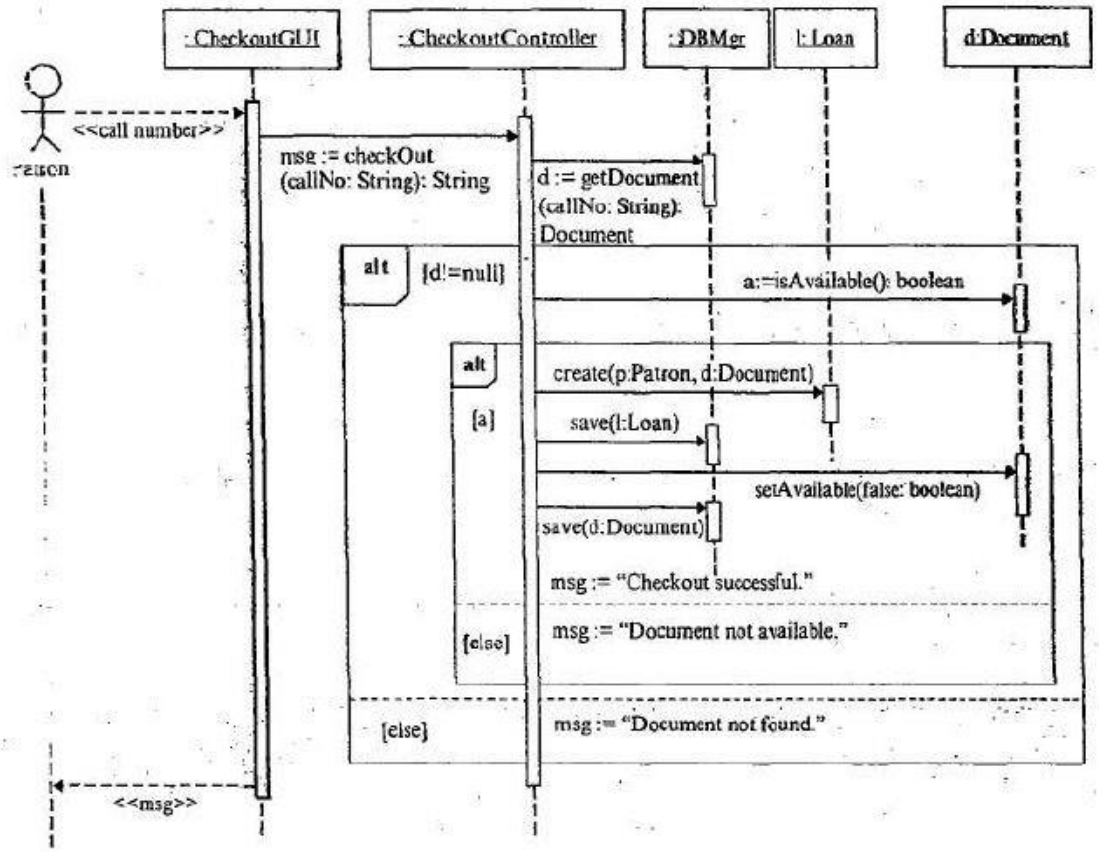
Types of Controller

There are two types of controllers:

(I) use Case Controller, and

(2) Facade Controller.

- A use case controller is responsible for handling all actor requests that are associated with a use case.
- The-controller in Figure was intended to be a use case controller for the Checkout use case.
- This is indicated by the name of the controller-that is, "Checkout Controller;" which implies that the controller is for the Checkout use case.



- If new requirements are added and new use cases are introduced, it only needs to add the corresponding use case controllers.
- Changes to existing requirements are limited to changing the relevant use case controllers and business objects.
- Information hiding is supported because changes to the business objects are shielded from the presentation, provided that the interfaces 'Of the controllers are kept stable.

Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/layers-of-osi-model/
https://www.youtube.com/watch?v=EzLMMsRR6Js

Important Books/Journals for further learning including the page nos.:
David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 PAGE NO:254-273

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : Object Oriented Software Engineering-16CSD04

Course Faculty :

Unit : III **Date of Lecture:**

Topic of Lecture: Creator patterns

Introduction : (Maximum 5 sentences) :

- Creator is a GRASP Pattern which helps to decide which class should be responsible for creating a new instance of a class.
- Object creation is an important process, and it is useful to have a principle in deciding who should create an instance of a class.
- These design patterns provide a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator.
- This gives program more flexibility in deciding which objects need to be created for a given use case.

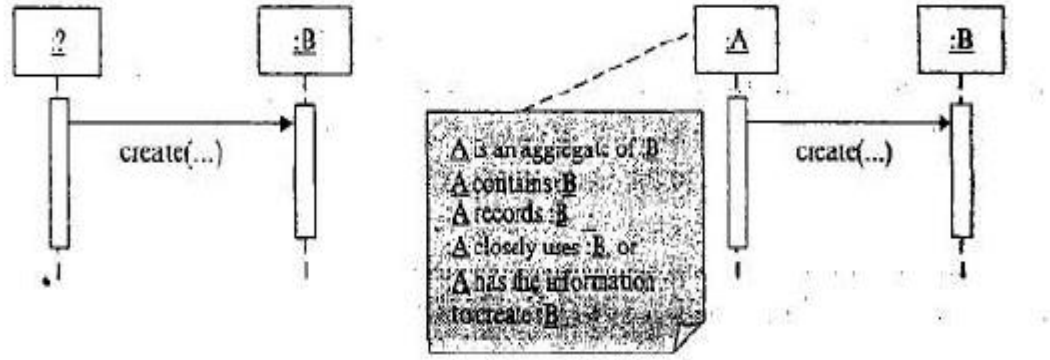
Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- Design Patterns
- Controller Pattern
- Assignment Pattern
- GUI
- Class and Objects

Detailed content of the Lecture:

What Is a Creator?

- Object creation is a common activity of an object-oriented system.
- Therefore, who should be assigned the responsibility to create an object deserves certain guidance. The creator pattern serves this purpose and is illustrated in below diagram.



(a) Who should be responsible for creating a B object?

(b) Creator suggests that :A should create :B

That is, the creator pattern suggests that the responsibility to create an object of Class B should be assigned to an object of Class A if one of the following holds:

ClassA is an aggregation of Class B.

- Since objects of Class A consist of objects of Class B, it is simple and straightforward to assign the creation responsibility to Class A. Because the dependency of Class A on Class B already exists, the call to the constructor of Class B does not introduce additional dependency. Reuse of
- Class A is easier than letting another class to create objects of Class R because this requires that the class be reused.

An object of ClassA contains objects of Class B.

- The fact that objects of Class A contain objects of Class B implies that the former may need to use or update the latter frequently.
- If this is the case, then letting objects of Class A create objects of Class B may result in a simple and easy-to-understand design.
- However, there are exceptions. For example, there are many cases where a container class is used only to store the elements, which are created by other classes.

An object of ClassA records objects of Class B.

- There are many cases where an object of Class A maintains objects of Class B.
- For example, a patient's medical file records the lab tests for the patient, the medical file check-in and checkout log records the check-in and checkout activities, and a purchase history records the details of each purchase item.
- In these and other similar cases, it may be simpler and more convenient to pass the required parameters to the medical file, the' log, and the purchase history, respectively, and let them create the elements.

An object of ClassA closely uses objects of ClassB

- There are many cases where an object of Class A closely uses objects of Class.
- These include, among others, an aggregate, container, or recording class that also retrieves or updates its elements frequently.

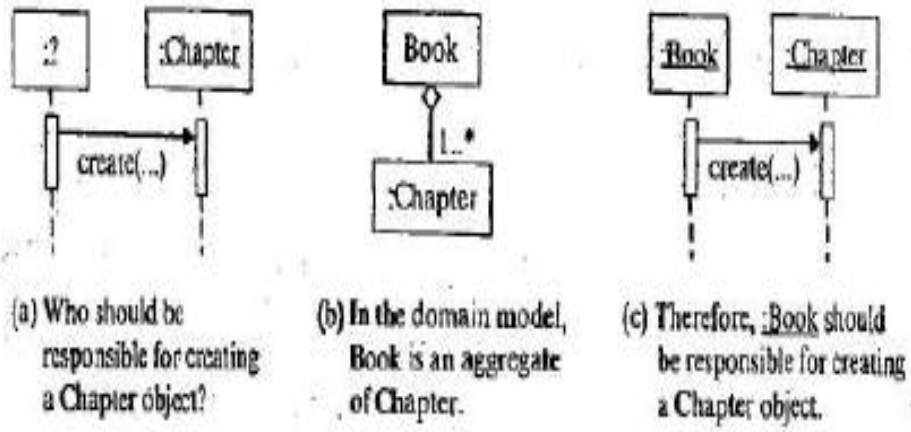
- An example is an inventory bookkeeper, which needs to check and update the inventory records frequently.
- Letting the inventory-bookkeeper create the inventory records simplifies the design and makes it easy to understand and reuse.

An object of Class A has the information to create objects of Class B.

- In many cases, input parameters must be passed to the constructor of a class.
- In such cases, it may be more convenient to let the object that has the input parameters to call the constructor if this does not deteriorate the cohesion of the creator.
- In diagram, the constructor of the Loan class requires a patron object and a document object.
- The checkout controller has these. Creating a Loan object is the responsibility of the checkout controller. Therefore, it is selected to create the Loan object.

Benefits of the Creator Pattern

- The creator pattern results in low coupling and better software reusability.
- The dependency of the creator on the object to be created already exists.
- Therefore, the creator pattern does not introduce additional dependency-that is, it results in low coupling.
- This also facilitates the reuse of the creator because the creator creates its dependent objects-in other words, there is no need to reuse anything else to create the dependent objects.

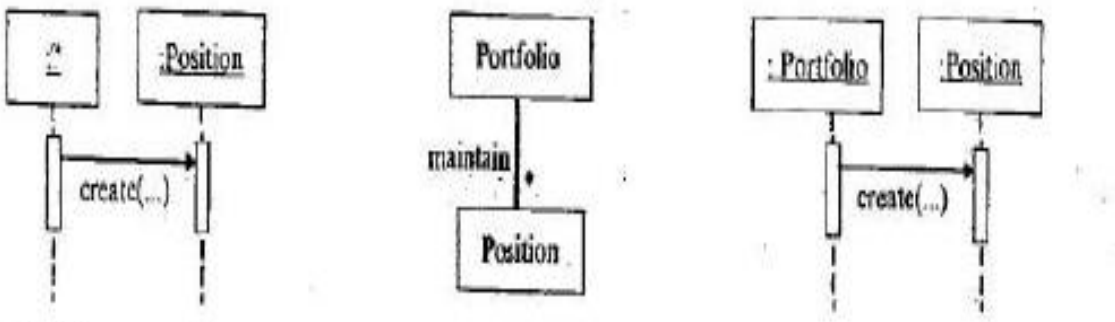


(a) Who should be responsible for creating a Chapter object?

(b) In the domain model, Book is an aggregate of Chapter.

(c) Therefore, :Book should be responsible for creating a Chapter object.

FIGURE 10.13 Who should create a Chapter object?



(a) Who should be responsible for creating a Position object?

(b) In the domain model, Portfolio maintains Positions.

(c) Therefore, :Portfolio should be responsible for creating a Position object.

When Does One Apply the Creator Pattern?

- The stages to apply the creator pattern are similar to the expert pattern.
- That is, depending on the development context, the pattern can be applied when the designer writes or modifies the use case scenario, or during the construction of the sequence diagram.

Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/layers-of-osi-model/
https://www.youtube.com/watch?v=EzLMMsRR6Js

Important Books/Journals for further learning including the page nos.:
David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 PAGE NO:270-274

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : Object Oriented Software Engineering-16CSD04

Course Faculty :

Unit : III **Date of Lecture:**

Topic of Lecture: Deriving a Design Class Diagram

Introduction : (Maximum 5 sentences) :

- A design class diagram (DCD) is an UMLClass diagram, derived from the behavioral models and the domain model. It serves as a design blueprint for test-driven development, integration testing, and maintenance.
- Package diagrams are useful for organizing and managing the Classes of a large DCD

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- UMLClass diagram
- Class diagram
- Package diagrams
- Behavioral model
- Domain Modeling

Introduction

- With the completion of interaction diagrams for use-case realizations, it is possible to identify the specification for the software classes (and interfaces) that participate in the software solution, and annotate them with design details, such as methods.

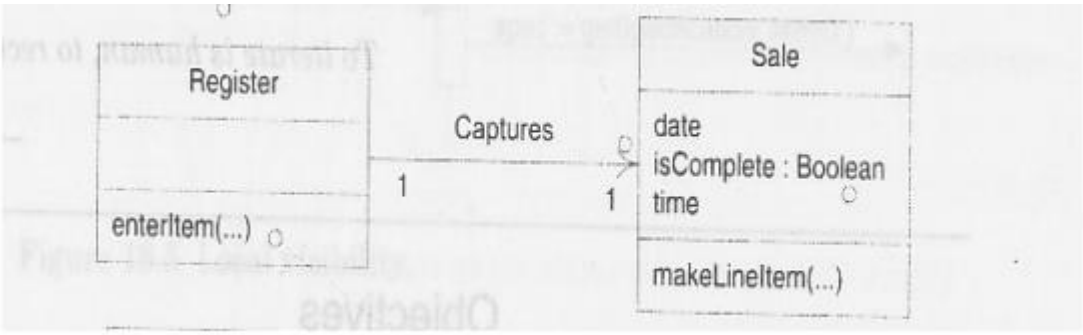
When to Create DCDs

- DCDs are usually created in parallel with interaction diagrams.
- Many classes, method names and relationships may be sketched out very early in design by applying responsibility assignment patterns, prior to the drawing of interaction diagrams.
- It is possible and desirable to do a little interaction diagramming, then update the DCDs, then extend the interaction diagrams some more, and so on.

Example DCD

- Here is an example of DCD with Register and Sale.

- The diagram consists of the methods of each class, attribute type information, and attribute visibility and navigation between objects along with the basic associations and attributes, which are created during Domain Modeling.



Terminology related to DCD

A design class diagram (DOD) illustrates the specifications for software classes and interfaces (for example, Java interfaces) in an application. Typical information includes;

- classes, associations and attributes
- interfaces, with their operations and constants
- methods
- attribute type information
- navigability
- dependencies

In contrast to conceptual classes in the Domain Model, design classes in the DCDs show definitions for software classes rather than real-world concepts.

Steps in creating DCD

- Identify software classes
- Illustrate them by Class Diagrams
- Add Method names
- Add Type information
- Add Association and Navigability
- Add Dependency relationship

Design Class Diagram Review Checklist

To ensure quality, the DCD must be reviewed by the team members using the following review checklist:

1. Ensure that the classes, attributes, operations, parameter types, return types, and relationships in the DCD are derived correctly according to the steps.

2. Does the DCD contain unnecessary classes, operations, .or relationships?

3. Does the naming of the classes, attributes, operations, and parameters communicate concisely the intended functionality and is it easy to understand?

4. Does the DCD clearly indicate the design patterns used? (This helps the programmer in the implementation phase.)

5. Compute metrics such as fan-in. fan-out, class size, depth in inheritance tree, and coupling between classes and identify potential problems.

ORGANIZE CLASSES WITH PACKAGE DIAGRAM

- The DCD may contain numerous classes, making it difficult to understand.
- In this case, UML package diagram is useful for organizing the classes into logical partitions called packages.
- The packages may be organized in different ways.

Commonly used organizations and their combination are:

1. Functional subsystem organization.
2. Architectural style organization.
3. Hybrid organization.

- The functional subsystem organization partitions the classes according to the functional subsystems of the software system.
- This results in packages that correspond to the functional subsystems of the software system.
- For example, the functional subsystems of a library information system include circulation subsystem, cataloguing subsystem, purchasing subsystem, interlibrary loan subsystem, and user assistance subsystem.
- Besides these, the system also has a persistence storage or database subsystem. Using this approach, six corresponding packages are defined.
- In addition to these packages, there is a package that contains classes belonging to the library information, system as a whole including Main GUI, Login and Logout GUl, and Configuration.

Video Content / Details of website for further learning (if any):
http://roshanchi.tripod.com/Documents/Study/OOAD/Notes/DCD.pdf
https://courses.cs.washington.edu/courses/cse403/11sp/lectures/lecture08-uml1.pdf

Important Books/Journals for further learning including the page nos.:
David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 PAGE NO:276-292

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : Object Oriented Software Engineering-16CSD04

Course Faculty :

Unit : III **Date of Lecture:**

Topic of Lecture: User Interface Design GUI widgets-process

Introduction : (Maximum 5 sentences) :

- User interface design is concerned with the design of the look and feel of the user interfaces.
- The design for change, separation of concerns, information-hiding, high-cohesion, low-coupling.
- Keep-it-simple-and-stupid software design principles should be applied during user interface design.
- The user interface of a software system is the means and mechanism through which a user interacts with the system to carry out business tasks.
- The users use the interface to request system services, provide user input, and receive system responses.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Graphical user interface
- Interface
- User Interface (UI) Design
- Widgets

Detailed content of the Lecture:

USER INTERFACE DESIGN :

- Several decades ago the user interface was extremely primitive.
- To input a program and feed data to it, one used a special typewriter to punch holes on a black tape, although others used punch cards.
- The punch tape was then mounted on an optical device, which read the tape and sent a bit stream to the computer.
- The output devices were the console typewriter and a line printer. Batch processing was the dominant mode of computing.
- In terms of today's standard, such a user interface is not user friendly at all, but at that time, it was

luxury to use a computer; therefore, nobody complained about it.

- Today's software systems offer graphical user interfaces (GUIs) and interactive mode of processing, although text-based interfaces are still used, Characteristics of GUIs include:

Window-based multitasking. Users can open multiple windows to work on and keep track of different tasks at the same time.

Easy to learn and use: Proper design of the look and feel using graphical widgets makes the user interface intuitive and easy to learn and use.

Multimedia presentation: The ability to use graphics, sound animation, and movies greatly enhances information presentation and communication.

WHY IS USER INTERFACE DESIGN IMPORTANT?

- Businesses and government organizations invest in computer hardware and software with the expectation to increase productivity and quality of service while lowering operating costs.
- The return on investment (ROI) depends on the design of the user interface because it is the sale communication channel between the user and the system.
- Through the user interface, users utilize the computer system to carry out business tasks. If the user interface is easy to use, then the user's productivity and work quality are increased. These in turn reduce operating costs.
- On the other hand, if the user interface is difficult to understand and use, then the users would avoid using the system, or their job error rates would be higher.
- Thus, the ROI is low. Unfortunately, the importance of user interface design is often underestimated.
- This results in products that offer excellent functionality and performance but do not sell because of poor user interfaces.

Graphical user interfaces widgets

- Graphical user interfaces are composed of GUI widgets, or simply widgets, such as windows, dialog boxes, menus, menu items, buttons and .many others.
- Different widgets serve different purposes, and proper use of the widgets is important.
- To save space and for simplicity, this section presents only widgets that are widely used and those used by user interfaces of stand-alone applications.
- Each platform has published its own user interface (UI) guidelines. These documents describe the design rules for software GUIs.

Container widgets

- Container widgets include window, dialog box, scroll pane, tabbed pane, and layered pane, among others. Windows are often used to represent the main display or main window of a stand-alone application or its subsystems.
- When the software system is started the main window is created and exists along with the application.

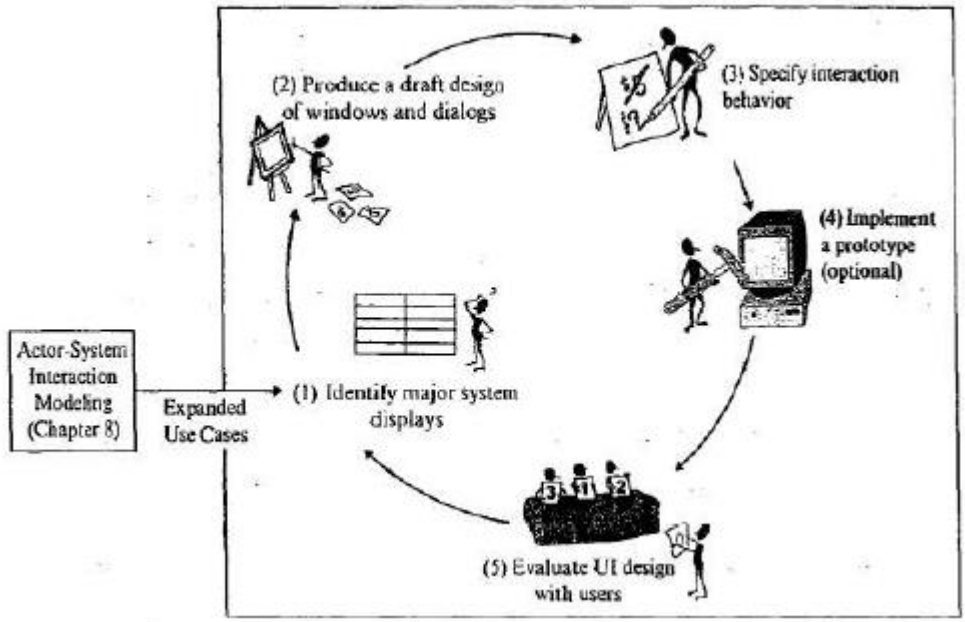
- It can be displayed anywhere in the user's desktop. Closing the window terminates the application and exiting the application closes the window.
- In Java, windows are decorated by frames to provide title bars, borders, and other window management buttons and menus.
- From a user's point of view, a window and a frame are not different and hence, these are not distinguished in this chapter.
- A dialog box is a window that is launched by another window to engage the user in a dialog. Closing a dialog box does not terminate the application software.

USER INTERFACE GUIDELINES

- UI guidelines are documents describing the rules for software GUI designs and operations. Applying these guidelines allows software developers and UI designers to realize software consistency within a platform.
- Many UI guidelines have been published, e.g., Microsoft Windows User Experience Interaction Guidelines, Mac OS X Human Interface Guidelines, and GNOME Human Interface Guidelines. International Conference Information Systems 2013297
- There are several benefits of adapting UI guidelines to GUIs.
- Applying UI guidelines allows consistent operations and designs of GUIs to be realized, which improves operational consistency between software products within a platform.
- A user can apply his or her experience of one software product to other software products without learning new operations, which increases operating efficiency.
- Additionally, software manuals in simple terms benefit both users and developers.
- Users can use the software products without reading the manual, while developers can decrease costs and burdens of preparing manuals because the operations are similar to other software products.
- Hence, using UI guidelines to make layout decisions simultaneously improves usability and reduces development costs.

USER INTERFACE DESIGN PROCESS :

- Below Figure shows the user interface design process.
- The input to the process is the expanded use cases produced in the current increment.
- The output is the user interface design.
- The steps of the process are outlined as follows and detailed in subsequent sections.



Step 1. Identifying major system displays.

- In this step, the system displays, user input and user actions are identified from the expanded use cases produced in the current iteration. These form the basis for the design of the look and feel in the next two steps.

Step 2. Producing a draft design of the windows and dialog boxes.

- In this step, a draft layout design of the windows and dialog boxes corresponding to the system displays is produced. This step designs the "look" of the user interface.

Step 3. Specifying interaction behavior.

- In this step, a state diagram is produced to specify the navigation relationships between the windows and dialog boxes. This step designs the "feel" of the user interface.

Step 4. Constructing a user interface prototype.

- This step is optional and produces a user interface prototype to show the look and feel as designed in the last two steps.

Step 5. Evaluating the design with users.

- In this step, the user interface design, and possibly the prototype, is presented to a group of user representatives to solicit their feedback, which is used to improve the design.

Video Content / Details of website for further learning (if any):
https://www.researchgate.net/publication/293090206_GUI_generation_based_on_user_interface_guidelines
https://www.interaction-design.org/literature/topics/ui-design

Important Books/Journals for further learning including the page nos.:
David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 PAGE NO:293-314

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : Object Oriented Software Engineering-16CSD04

Course Faculty :

Unit : IV **Date of Lecture:**

Topic of Lecture: Implementation Considerations

Introduction : (Maximum 5 sentences) :

- Everyone in the team should follow the same coding standards.
- Test-driven development, pair programming, and code review improve the quality of the code.
- Classes should be implemented according to their dependencies to reduce the need for test stubs.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

Coding Standards
Coding conventions
Organizing the implementation Artifacts
Generating code from design
Pair Programming

Detailed content of the Lecture:

Coding Standards

Coding standards usually include the following major items:

File header.

Many companies' coding standards require a file header at the beginning of each program file. It specifies the file location. version number, author, project, update history, and other useful information.

Description of classes.

Many companies also require a functional description for each class. It includes:

a. Purpose-a statement of the purpose of the class.

b. Description of methods--a brief description of the purpose of each method, the parameters, returns type and other input and output such as files. Database tables and text fields accessed and updated by the method, The list should not include the ordinary get and set functions.

Description of fields

A brief description of each field including the name of the field, data type, range of values, initialization requirement, and values of significance.

Coding conventions include:

Naming conventions.

These conventions specify rules for naming packages, modules, paths, Files, classes, attributes, functions, constants and the like. Naming conventions should help program understanding and maintenance.

Formatting conventions.

Formatting conventions specify formatting rules used to arrange program statements. These include line break, indentation, alignment and spacing

In-code commend conventions.

If it is written properly, in-code comments facilitate program understanding and maintenance.

ORGANIZING THE IMPLEMENTATION ARTIFACTS

Architectural-style organization.

This approach organizes the classes accordingto the building blocks of the software architecture. It Shows the correspondence between the N-tier architecture and the packages forthe library information system discussed

Functional subsystem organization.

This approach organizes the classes according to the bnctional subsystems of the software system.

Hybrid organization.

This approach combines the architectural-style organization and the functional subsystem organization. Two approaches exist: the architectural-style functional subsystem organization and the functional subsystem architectural-style organization.

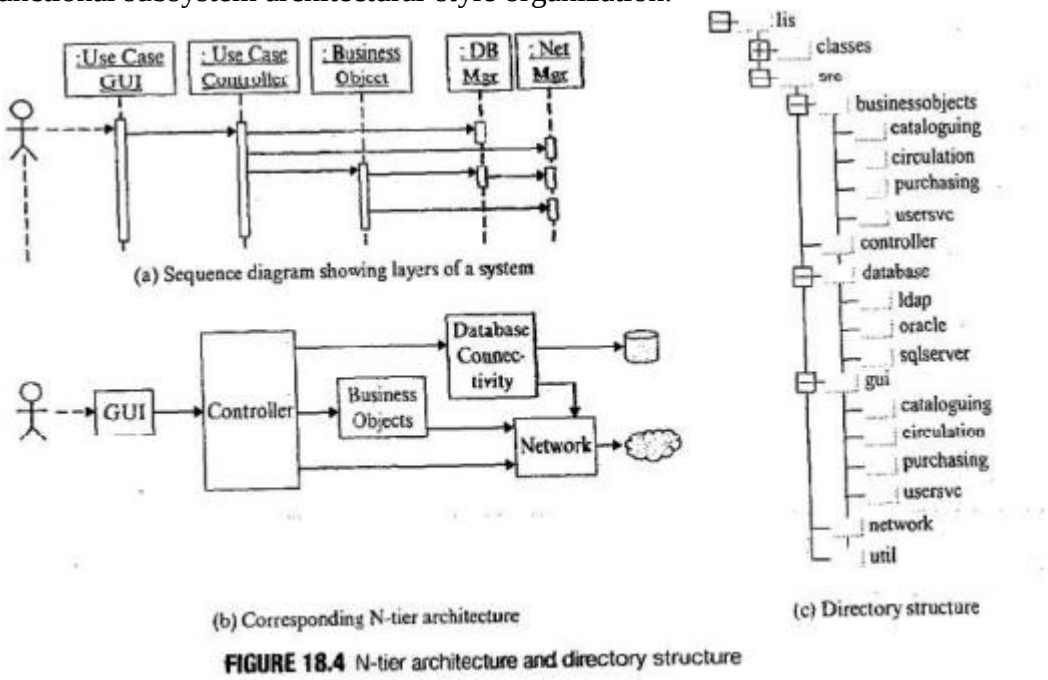


FIGURE 18.4 N-tier architecture and directory structure

GENERATING CODE FROM DESIGN

From Sequence Diagram to Method Code Skeleton

Sequence diagrams model the behavioral aspect of objects. The Long narrow rectangle under the Checkout Controller object indicates the execution of the checkout (callNo: String):String method of the Checkout Controller.

The arrow lines that go out from this rectangle represent calls to functions of other objects. The large box with "alt" at the upper-left corner implies a selection or if-then-else statement.

Implementing Association Relationship

One-to-one association.

A one-to-one association between class A and class B is implemented by A holding a reference to B if A calls a function of B, and/or by B holding a reference to A if B calls a function of A.

One-to-many association.

A one-to-many association between class A and class B is implemented by A holding a collection of references to B if A calls the functions of B instances, or by B holding a reference to A if instances of B call a function of A.

Many-to-many association.

A many-to-many association between class A and class B is similarly implemented by a collection of references from A to B, and vice versa.

PAIR PROGRAMMING

Pair programming is an emerging programming technique that requires two people to program together at one machine, with one keyboard and one mouse.

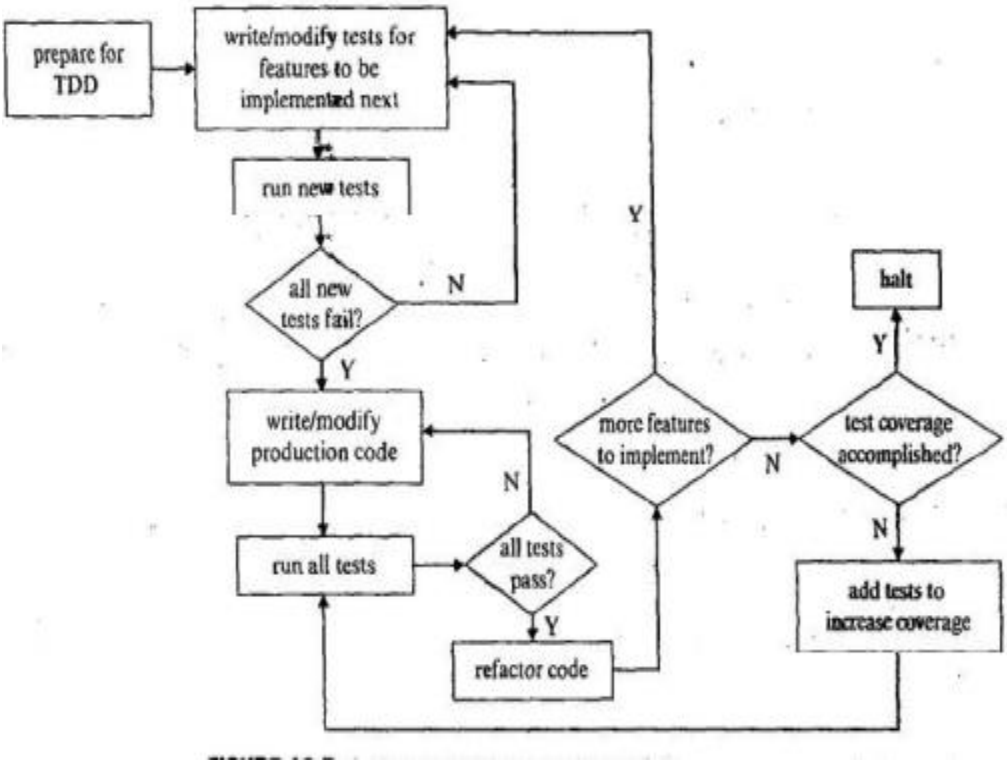
The two programmers play two different roles but both work on the same program simultaneously.

While the one with the keyboard and the mouse focuses on the best way to implement the functionality the other reviews the program as it is being typed. For convenience, these two roles are referred to as the writer and the reviewer.

TEST-DRIVEN DEVELOPMENT

WORKFLOW

- Prepare for Test Driven Development
- Write Tests
- Implement and Test the Features
- Repeat until all features are correctly implemented
- Accomplish test coverage



Merits of Test-Driven Development

- TDD requires the programmer to understand the functionality and implement testable features. Testability is an important attribute of software, especially software requirements.
- TDD constantly validates the implementation with respect to the tests. It helps the team detect and remove defects. As a result, TOO produces high-quality code.
- TDD focuses on the desired functionality first but also addresses the other quality aspects such as program structure and readability through refactoring.
- TDD facilitates debugging because incremental implementation of the features makes it easy to locate and fix errors.

Potential Problems

The test cases may be too weak to ensure that the program indeed correctly implements the desired functionality.

The test cases may be too focused on the main functionality and overlook other cases that may cause the program to crash or behave incorrectly.

The test cases or test scripts are themselves programs. If they are not written in accordance to coding standards and conventions then the maintenance of these programs is a nightmare.

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

https://slideplayer.com/slide/9295789/
https://slideplayer.com/slide/6420428/

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

David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 Page NO:450-467

Journals

https://www.researchgate.net/publication/200484093_
https://www.clutejournals.com/index.php/RBIS/article/download/4482/4570

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **IV** **Date of Lecture:**

Topic of Lecture: SOFTWARE QUALITY AND METRICS

Introduction : (Maximum 5 sentences) :

- Software quality assurance encompasses a set of activities to ensure that the software under development or modification will meet functional and quality requirements.
- Software quality assurance activities are life-cycle activities.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- Benefits of Software quality Assurance
- Quality Measurements and Metrics
- Usefulness Of Quality Measurements And Metrics
- Conventional Quality Metrics

Detailed content of the Lecture:

BENEFITS OF SOFTWARE QUALITY ASSURANCE

- The importance of software quality can never be overstated. Our society depends on software to perform almost everything.
- Software bugs cost billions of dollars owing to property damages, productivity losses, bodily injuries, and loss of lives.
- One of the worst software bugs of all time is the Therac-25 bug, which caused the radiation.
- Qualitative formulation as such is ambiguous because different persons can interpret the same requirement-differently.

QUALITY MEASUREMENTS AND METRICS

Software quality attributes state the important aspects of software. However, they are at most qualitative assessments of the software. The qualitative nature allows subjective evaluations.

- A software measurement is an objective and quantitative assessment of software attribute.
- Software metric is a standard software measurement.
- An indicator is a measurement or metric value, believed to have a significant implication.

USEFULNESS OF QUALITY MEASUREMENTS AND METRICS

- Definition and use of Indicators
- Directing valuable resources to critical areas
- Quantitative Comparison of similar projects and systems
- Quantitative assessment of improvement
- Quantitative assessment of Technology
- Quantitative assessment of process improvement

Conventional Quality Metrics

Requirements Metrics

f(state, stimulus) →(state, response)

Design Metrics

The module design complexity mdc(M) is calculated by: .

Mdc[M] == d+ I

That is, the integration complexity is the number of atomic binary decisions plus one. Using this formula to compute the integration complexity for the MO module results in

$SI(MO) = N_{abd} + 1 = 3 + 1 = 4$

Implementation and System metrics

The reliability metric uses the mean time between failures (MTBF) as the measurement. It is calculated as the sum of the mean time to failure (MTTF) and the mean time to repair (MTTR) of the system. That is, reliability is:

MTBF = MTIF + MTTR

Object-Oriented Quality Metrics

The object-oriented paradigm introduces a number of powerful features such as encapsulation, inheritance, polymorphism, and dynamic binding. Accordingly. there are a number of object-oriented measurements and metrics:

Weighted Methods per Class (WMC).

The WMC for a class C is the sum of the complexity metrics of the methods of a class. It is computed as

WMC(C) = Cml + C m 2 +-... +c mn

where i = 1,2, ... n, are - the complexity metrics of the methods of C.

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

https://www.youtube.com/watch?v=M7ZVcQOSVF4

https://www.youtube.com/watch?v=5_cTi5xBlYg

https://www.geeksforgeeks.org/software-engineering-software-quality-assurance/

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

David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 Page No:470-482

Journals:

https://ieeexplore.ieee.org/document/5010196

https://www.springer.com/journal/11219

Course Faculty

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

L-

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code **:Object Oriented Software Engineering-16CSD04**

Course Faculty **:**

Unit **: IVDate of Lecture:**

Topic of Lecture: Software Verificationand Validation Techniques

Introduction : (Maximum 5 sentences)

- Verification and validation are SQA activities to ensure that the software process and product confirm to established quality requirements.
- Software verification and validation are important because software is used inall sectors of our society.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Inspection
- Walkthrough
- Peer Review
- Verification and Validation Life Cycle

Detailed content of the Lecture:

Inspection

Inspection checks the product against a list of common errors, anomalies and non-Compliances to standards and conventions. It is similar to car inspection that is required in many countries.

• Use of uninitialized variables objects, references, or pointers.
• Calling the wrong polymorphic function.
• Incorrect function invocation for example, incorrect parameters are passed to the function or the parameters are misplaced.
• Non terminating loops or incorrect loop termination conditions.
• Mismatch in array dimensions, causing an array index out-of-bounds exception.
• Uncaught/unhandled runtime exceptions.

- Incorrect business logic
- Inconsistent business logic
- Incomplete business logic

WALKTHROUGH

Walkthrough manually executes the product using simple test data. Usually, the developer who produces the product leads the team to perform the walkthrough The team checks the product step by step while reading aloud.

- Applicability
- Effectiveness
- Participants
- Procedure
    1. A product overview is presented to the participants if desired.
    2. The developer loudly reads through the product and provides necessary explanations. The other team members ask questions and raise doubts.

3. The developer fixes the problems, produces a summary list, and obtains approval from the participants.

PEER REVIEW

In peer review, the product is reviewed by peers, who are guided by a list of review questions, designed to qualitatively assess aspects of the product.

The reviewer's assessments of the product may vary drastically because the assessments are heavily influenced by the reviewer's knowledge, experience, background, and criticality**.**

VERIFICATION AND VALIDATION IN THE LIFE CYCLE

Verification and Validation in the Requirements Phase

Verification and validation in the requirements phase aims at detecting errors and anomalies in the requirements specification and the analysis models including the domain model and use case diagrams.

- Completeness
- Consistency
- Unambiguity
- Traceability
- Feasibility

Verification and Validation in the Design Phase

Checking the correctness of the design is a validation activity to ensure that the design corresponds to the real needs of the customer.

If the requirements and constraints correctly and adequately specify the real needs of the customer, then assessing the satisfiability of the design with respect to the requirements and constraints is a verification approach to ensure that the design corresponds the customer's real needs.

Verification and Validation in the implementation Phase

Inspection and peer review in the implementation phase are aimed to ensure the following:

• The implemented interfaces and interaction behavior between the various components are consistent

• The source code satisfies the organization's coding standards and quality, metrics such as information hiding high cohesion low coupling, and acceptable cyclomatic complexity, for example not to exceed 10.

• The programming constructs of the implementation language are used properly.

Video Content / Details of website for further learning (if any):
Can be added as link
https://www.geeksforgeeks.org/software-engineering-verification-and-validation/
https://www.youtube.com/watch?v=qxitgylm1EU

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

David Kung Object-Oriented Software Engineering:An Agile Unified Methodology McGraw-Hill Education 2013Page No:483-490

Jorunals:
https://www.springer.com/journal/11219/updates/17193268

Course Faculty

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

LECTURE HANDOUTS

CSE | **II/IV**

Course Name with Code :**Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **IVDate of Lecture:**

Topic of Lecture:SQA Functions

Introduction : (Maximum 5 sentences)

- The overall objective of SQA is to ensure that the software development process is carried out as required, and the software system meets the requirements and quality standards.
- SQA, as an area of the software engineering discipline, is responsible for the research, development and validation of cost-effective processes, methods, and tools to accomplish these goals.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Definition of Process and Standards
- Definition of Process and Methodology
- Definition of Metrics and Indicators
- Quality Management
- Process Improvement:

Detailed content of the Lecture:

Definition of Process and Standards

The definition of processes and standards function is responsible for developing and defining a framework for ensuring software quality for the whole organization.

- Definition of software development, and quality management processes and methodologies.
- Definition of SQA standards, procedures, and guidelines for carrying out the SQA activities during the life cycle.
- Definition of quality metrics and indicators for 'quality measurement and assessment.

Definition of Process and Methodology

The importance of a software development process and a development methodology is discussed in Chapter 2, where several software process models are described. A software development methodology implements a software process.

Definition of SQA Standards and Procedures

Another responsibility of the SQA component IS defining quality standards and procedures for all software projects to comply. These include process standards and product standards.

The process standards define the requirements on the development processes and methodologies.

Definition of Metrics and Indicators

- Software quality assurance requires measurements so that quality can be assessed. Metrics and indicators are needed for measuring and assessing software quality.
- This function-of the SQA component identifies and defines the metrics to be used to measure the process and product aspects of the projects in the organization.

Quality Management

The definition of processes and standards function defines the SQA framework for the organization.

It consists of two activities:

1. **Quality planning.** This activity takes place at the beginning of each project. It produces a quality plan for the specific project.
    - Purpose
    - Management
    - Standards and Conventions
    - Review and Audits
    - Configuration Management
    - Process Methodologies, tools and techniques
    - Metrics and indicators
2. **Quality control.** This activity takes place throughout the entire project. It monitors the execution of the quality plan as well as modifies the quality plan to respond to changes in the reality.

SQA Control

- This SQA function ensures that the SQA plan is carried out correctly. SQA training could be one of the important activities of this function .
- The training is aimed to educate the developers of the importance of SQA, the organization's SQA standards and procedures, the available SQA tools as well as how to use the tools to perform SQA activities.

Process Improvement:

- Defining metrics and data collection methods.
- Collecting data for measuring the process
- Calculating the metrics and indicators.
- Recommending improvement actions.

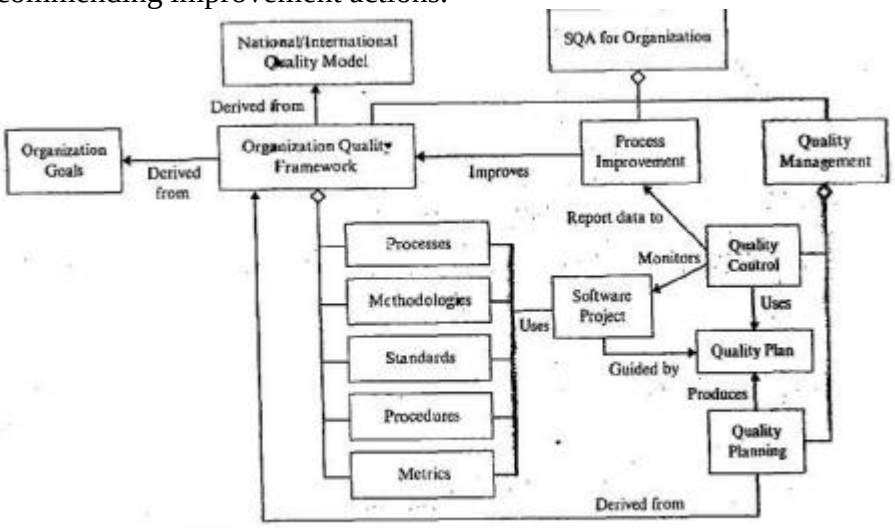


FIGURE 19.6 A domain model for SQA functions

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

https://www.tutorialspoint.com/software_quality_management_sqa_components.htm
https://www.youtube.com/watch?v=cCzh9kSiRCo

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

David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 Page NO:486-504

Journals

http://ijarcsms.com/docs/paper/volume2/issue3/V2I3-0131.pdf

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **IV** **Date of Lecture:**

Topic of Lecture: Black Box And White Box Testing

Introduction : (Maximum 5 sentences) :

- This section presents some of the well-known conventional black box testing techniques, which can be used to test the member functions of a class and provide a basis for learning object-oriented testing methods.
- Unlike black-box testing techniques, white-box testing derives test cases from the internal structure or logic of the CUT. Most white-box approaches generate test cases through the analysis of the source code.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- Equivalence Partitioning
- Boundary value analysis
- Cause effect analysis

Detailed content of the Lecture:

CONVENTIONAL BLACK BOX TESTING

- Functional testing derives test cases from the requirements, or functional specification of the component under test (CUT).
- Functional testing is also referred to as black-box testing because it treats the CUT as a black box. In illustration, consider a purge function that eliminates duplicate elements from an integer list.

The input to the purge function is a list of elements, denoted L = A I, A2, ... , An. The output of the function is a list of elements L' ~ AI', A.2', Am', m <= n.
The output list must not contain duplicate elements.
This can be specified mathematically as:

(l)(Vi')(Vj'((i', j' <= rz) A (Ai' = Aj') -+ (i' = j'))

This formula facilitates the derivation of test cases.

There are three commonly used black-box testing techniques.
These are

- equivalence partitioning
- boundary value analysis
- cause-effect analysis.

EQUIVALENCE PARTITIONING

Equivalence partitioning divides the input and output domains into a number of disjoint subsets, and selects one test case from each of these disjoint subsets.

The key to equivalence partitioning is to identify an equivalence relation among the elements of

an input domain. An equivalence relation is a reflexive, symmetric, and transitive relation.

BOUNDARY VALUE ANALYSIS

Equivalence partitioning divides all possible input or output values into equivalence classes and selects test cases from each of the partitions.

The boundary value analysis selects test cases at and near the boundaries of the equivalence classes. Therefore, the two test case generation methods complement each other.

CAUSE-EFFECT ANALYSIS

- Cause-effect analysis is similar to the functional test example described , except that a decision table is constructed to help the generation of the test cases.
- First, the dependencies between the input variables and the outcome of the CUT are identified.
- Second, values for the input variables are determined.
- Third a decision table is constructed to show the correspondence between the input value combinations and the outcome of the CUT.
- Finally, test cases are derived from the rules of the decision table.

CONVENTIONAL WHITE BOX TESTING

Basis Path Testing

Basis path testing generates test cases to exercise the independent control flow paths, called basis paths, of the CUT. The basis paths are derived from the CUT's flow graph, which is constructed using a number of flow graph notations.

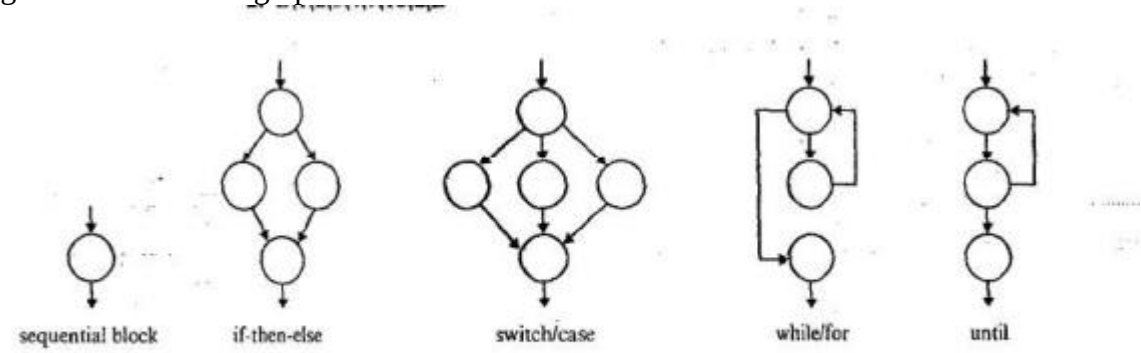


FIGURE 20.5 Flow graph notation and a flow graph

Cyclomatic Complexity

The number of basis paths of the CUT is defined as the cyclomatic complexity of the CUT.lt is determined in three equivalent ways. That is, either of these three approaches can be used to determine the cyclomatic complexity:

1. **Number of closed regions plus one.** This approach obtains the cyclomatic complexity by adding one to the number of closed regions in the How graph. In there are three such regions; therefore, the cyclomatic complex-

2**. Number of nodes and edges**. In this approach, the cyclomatic complexity is the Number of edges minus the number of nodes plus 2. In Figure 20.6(b), there are 14edges and 12 nodes therefore; the cyclomatic complexity is 14 - 12 + 2 = 4.

3. **Number of atomic binary conditions plus one.** The cyclomatic complexity is the number of atomic binary conditions plus I, there are three atomic binary conditions. Therefore, thecyc1omatic-complexity is 4. When using this approach, treat each n-ary condition as n - 1 binary condition.

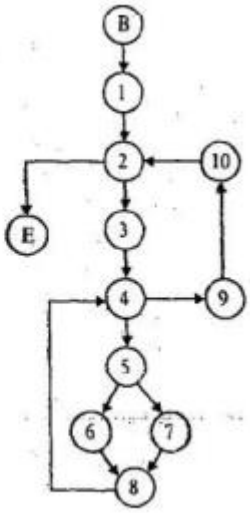
Flow Graph Test Coverage Criteria

Flow graph-based test coverage criteria are defined as follows, ranging from the weakest to the strongest-The weakest criterion yields the Lowest confidence on the CUT, while the strongest criterion provides the highest confidence on the CUT:

1. Node Coverage
2. Edge Coverage
3. Basis path coverage

```
public void purge(LinkedList list) {
  Item p, q;
(1)  p=(Item)list.getFirst();
(2)  while (p != null) {
(3)    q=(Item)p.getNext();
(4)    while (q !=null) {
(5)      if (p.compareTo(q)==0)
(6)        list.remove (q);
(7)      else q=(Item)q. getNext();
(8)    }
(9)    p=(Item)p. getNext();
(10)  }
(11) }
```

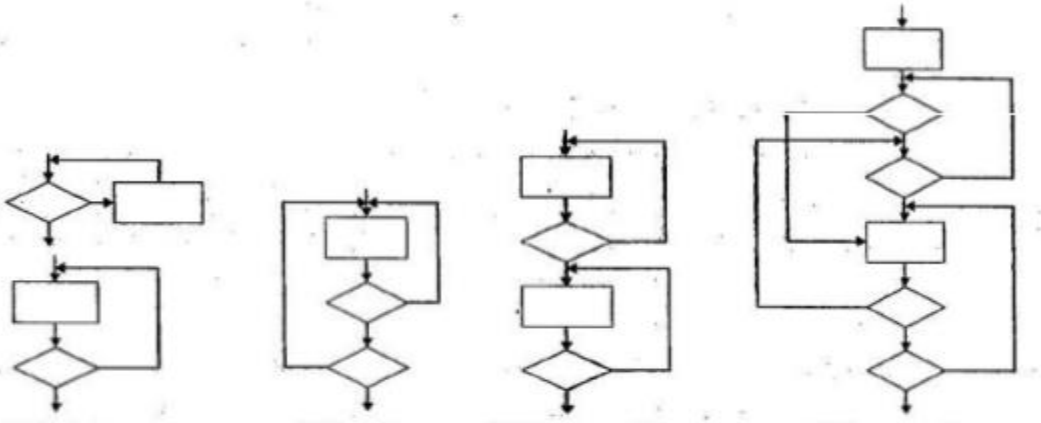
(a) The purge function



(b) Flow graph for the purge program

Testing loops

All path coverage is practically impossible because many programs contain nested loops that iterate numerous numbers of times, resulting in countless numbers of paths.



(a) Simple loops (b) Nested loops (c) Concatenated loops (d) Unstructured loops

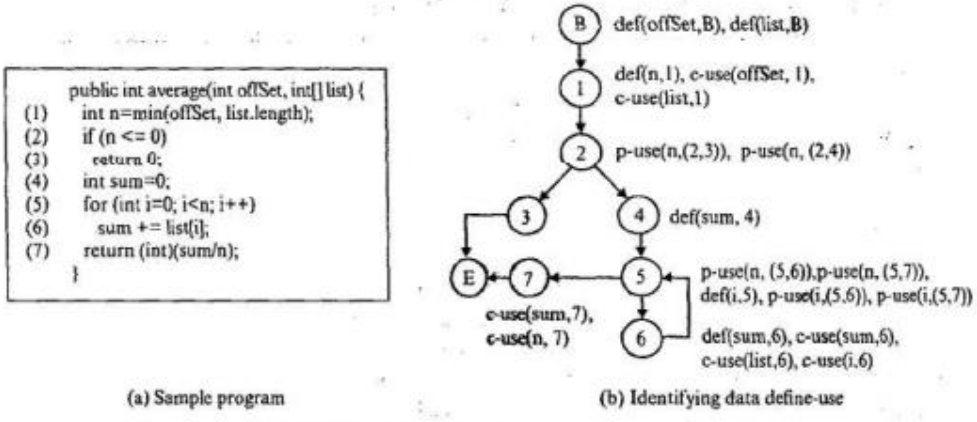
Data Flow Testing

Data flow testing focuses on the define-use relationships of selected program variables. A variable is defined if its value is updated at a program location.

A variable can be used to evaluate a condition or compute a value. These are called predicate use or p-use, and computation use or C-use, respectively.

Interprocedural Data Flow Testing

- The last section presents test case generation based on data define-use paths within a function.
- This has been called intraprocedural data flow testing. There are many cases in which a variable is defined in one function and used in another function, resulting in interprocedural data flow testing.



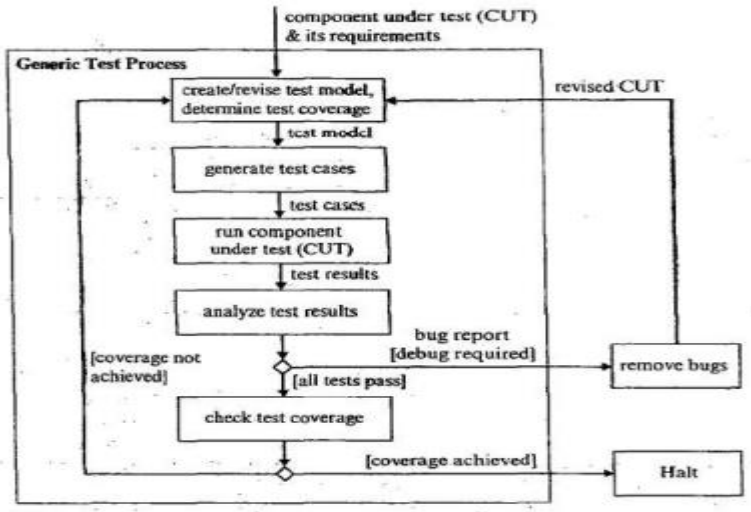
(a) Sample program (b) Identifying data define-use

Interprocedural Data Flow Testing

- The last section presents test case generation based on data define-use paths within a function.
- This has been called intraprocedural data flow testing. There are many cases in which a variable is defined in one function and used in another function, resulting in interprocedural data flow testing.

TEST COVERAGE

The notion of test coverage is mentioned a few times during the presentation of the test methods in previous sections. It was but defined earlier because the term is somewhat abstract.



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

https://www.youtube.com/watch?v=Wi75S5TTfQ0
https://www.geeksforgeeks.org/software-engineering-black-box-testing/

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

David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 Page No:504-515

Journals:

https://www.researchgate.net/publication/276198111
https://www.ijarcst.com/doc/vol2-issue3/ver.1/nidhi_gupta.pdf

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **IV** **Date of Lecture:**

Topic of Lecture: OO Software Testing

Introduction : (Maximum 5 sentences)
These techniques are applicable to testing the methods of a class.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- Use case based Testing
- Object state testing
- Generate test cases
- Testing Class hierarchy

Detailed content of the Lecture:

Use Case-Based Testing

Use case-based testing, as its name suggests, derives test cases from the use case specifications**.**

Step l. Identify actor input and actor actions.
Step 2. Determine input values.
Valid Input
Invalid Input
Exceptional Cases
Step 3. Generate test cases.
Step 4. Generate concrete tests.
Step 5. Implement and run the tests.

Object State Testing with Class Bench

Many objects exhibit state-dependent behavior. or state behavior, for short. For example, pushing an element onto a stack may increment the size of the stack or cause a stack full exception.

The Test Model and Test Coverage

The Class Bench test model is a finite state machine, where the states represent the states of the CUT and the transitions represent executions of functions of the CUT.

Three test coverage criteria can be defined, in ascending order of rigidness:

Node Coverage
Edge Coverage
Path Coverage

Generate Test Cases

Each test case is a path of the test model; beginning from an initial state and ending at some state.

Implement and Run the Test Cases

The next step is to implement the test cases in l Unit and run the test cases using one of the J Unit test runners**.**

Using a Test Oracle

A test oracle is a piece of software that simulates the functionality and behavior of a CUT to facilitate checking of the test result produced by the CUT.

Testing Class Hierarchy

Inheritance is a unique feature of object-oriented programs. Testing an inheritance hierarchy should begin with the root class and work downwards.

Conventional black-box and white-box test methods can be used to test the functions of a class, and the Class Bench approach can be applied to testing object stale behavior.

Testing Exception-Handling Capabilities

Exception handling is an important feature of object-oriented programming. Therefore, the exception-handling capability of the CUT should be tested.

1. The CUT throws Exceptions.

The CUT does not throw a potential exception.

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

https://www.geeksforgeeks.org/object-oriented-testing-in-software-testing/
https://www.youtube.com/watch?v=ssJZQf3kQcQ

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

David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 Page No:518-524

Journals:

http://ijcsit.com/docs/Volume%202/vol2issue5/ijcsit2011020571.pdf

Course Faculty

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-

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **IV** **Date of Lecture:**

Topic of Lecture: Test Web Application Testing For Functional Requirements

Introduction : (Maximum 5 sentences) :

- Many web applications use objects to implement the software running in the server side. Therefore, teams that develop object-oriented applications must know how to test web applications.

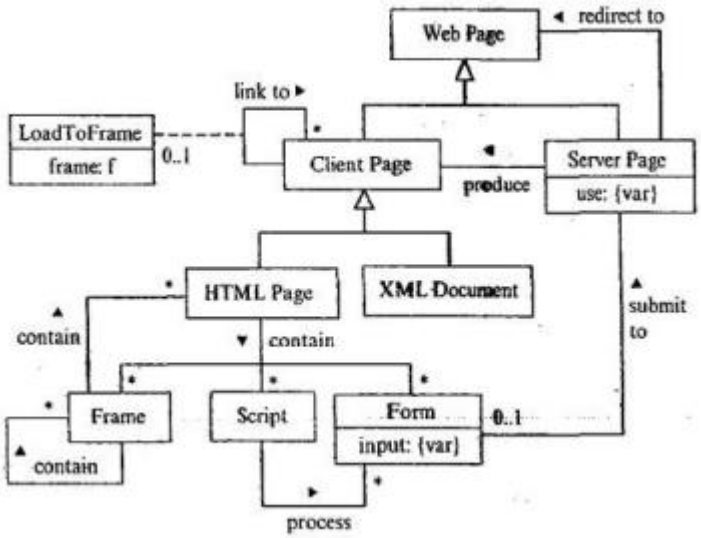
Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- Web Application testing
- Test case generation
- Non functional requirements

Detailed content of the Lecture:

Object-Oriented Model for Web Application Testing

A web application involves multiple types of documents that relate to each other in a complex manner. To help understand the documents and their relationships. A test model is constructed. The model is useful for deriving test cases.



Static Analysis Using the Object-Oriented Model

The test model shown can be used to detect a number of anomalies. For example, pages that exist on the web server but do not appear in the test model are unreachable pages.

Test Case Generation Using the Object-Oriented Model

Several test methods presented previously can be used to test the Java bean classes and the JSP pages.

Partition testing divides the domains ofthe form's input variables and selects test data from the partitions to form tests. Boundary testing selects test data at the boundaries of the partitions.

Web Application Testing with Http Unit

The test cases generated in the last section can be implemented and executed using the Http Unit open source software: Http unit n is an extension of Unit to web application testing. It emulates browser behavior and allows a test case to send requests to and receive responses from the web server.

TESTING FOR NONFUNCTIONAL REQUIREMENTS

Software systems must also be tested with respect to nonfunctional requirements. These include performance testing, stress testing, and security testing, among others**.**

Performance and Stress testing

Performance testing is aimed to assess several aspects of the software or system, including system workload, throughput, response time, efficiency, and resource utilization. Workload and throughput measure the amount of work that the system processes and produces.

Testing for Security

Conventional test methods and techniques are aimed at detecting errors in the software while demonstrating that the software accomplishes its intended functionality and behavior.

White-box testing approaches are applied to detect security vulnerabilities and generate test cases.

Testing User Interface

- Defects in the look and feel of the user interface.
- Defects in data entry and output display.
- Defects in the actor-system interaction behavior.
- Defects in error handling.
- Defects in documentation and help facility.

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

https://blog.stackpath.com/web-application/

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

David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 page No:529-530

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **IV** **Date of Lecture:**

Topic of Lecture: Testing Life Cycle

Introduction : (Maximum 5 sentences) :

- Software testing is a life-cycle activity, meaning that it should be taken into consideration in each of the life-cycle phases. The traditional life-cycle activities and their relationships to testing are illustrated using a V-shape diagram.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- Testing Objectives
- Integration testing

Detailed content of the Lecture:

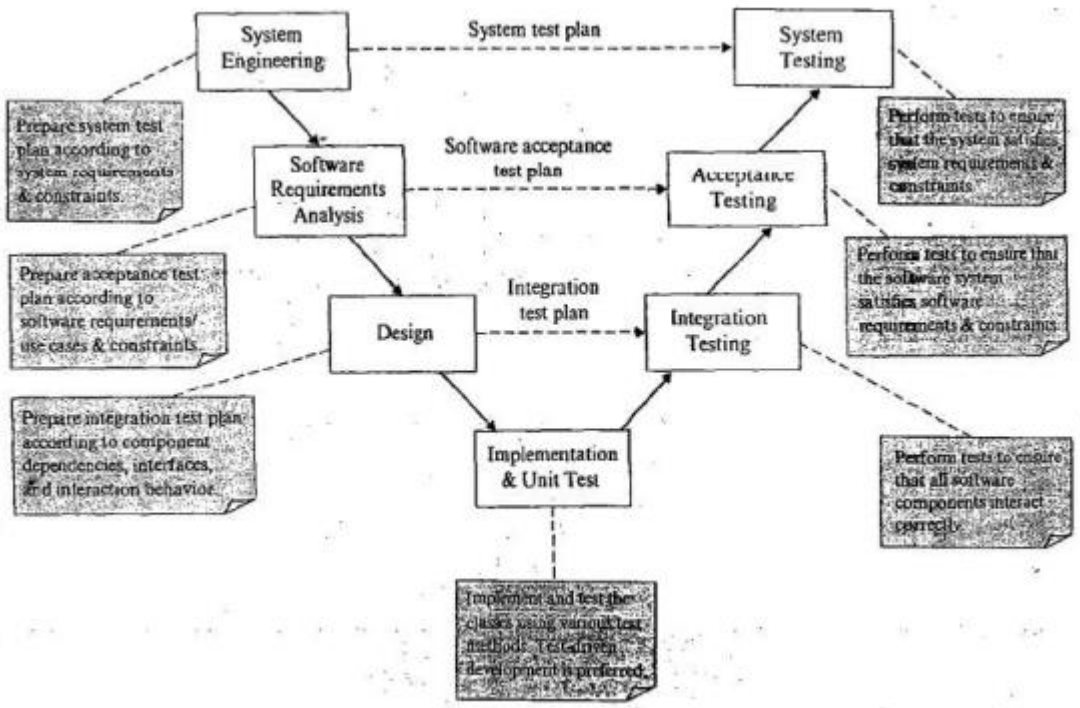
- The left leg of the V shape is mainly concerned with the construction activities of the system.
- During this period, only static testing are possible and are performed using inspection, walkthrough, and peer reviews.

A test Plan generally specifies the following items

- Test Objectives
- Types of test
- Test methods and techniques
- Test cases
- Test Coverage Criteria
- Documents Needed
- Required resources
- Effort estimation schedule

Traditionally, integration and integration testing are performed using the following strategies:

- Big bang.
- Top down Integration
- Bottom up Integration
- Critical/ high priority Components
- Available Components



If the functions and interfaces of the individual components are implemented according to the design specification, then integration testing should proceed relatively smoothly.

During the implementation and unit testing phase, the software components are implemented and tested by the individual developers.

It requires the programmer to understand the functionality prior to implementing the functionality.

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

https://www.guru99.com/software-testing-life-cycle.html

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

David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 page No:529-532

Journals

https://www.irjet.net/archives/V6/i1/IRJET-V6I1234.pdf

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **IV** **Date of Lecture:**

Topic of Lecture: Regression Testing

Introduction : (Maximum 5 sentences) :

- Changing a software system or its components is inevitable. This takes place during the development as well as the maintenance phases.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- Importance of Regression testing

Detailed content of the Lecture:

- Change alters the functionality, behavior, and performance. Therefore, retesting is required to ensure that the system or its components still satisfy the functional, performance, and security requirements.
- Often, regression testing executes all the existing test cases or a selected subset to ensure that the software system or its components pass the tests.
- Selecting a subset of the existing test cases can save time and effort. Test cases are selected according to the components that are changed or affected by the changes.
- Tools for selecting regression test cases have been developed. Some of the tools instrument and execute the software before making changes.
- This information along with the changed and affected classes are used to select the test cases that need to be rerun.
- If X Unit has been used during development testing then regression testing simply reruns the X Unit test cases.
- However, X Unit test cases usually do not include system testing and user interface testing. In these cases, other regression testing tools, such as Win Runner and VI Gestures Collector (a plug-in of Net Beans) should be used.
- These tools record the user actions and play back the recorded test scripts during regression testing.

Software testing is costly and time consuming. Therefore, it needs to know how much Testing is adequate, or when to stop testing.

Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/software-engineering-regression-testing/

Important Books/Journals for further learning including the page nos.:
David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 Page No:532-533
Journals:
https://www.springer.com/journal/10921

Course Faculty

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

LECTURE HANDOUTS

CSE | **II/IV**

Course Name with Code :**Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **VDate of Lecture:**

Topic of Lecture: Software maintenance

Introduction : (Maximum 5 sentences)

- Software maintenance is modifying a software system or component after delivery to correct faults, improve performance, add new capabilities, or adapt to a changed environment. (IEEE Standard 610.12-1991)
- Software maintenance consumes 600/0-80%of the aotal life-cycle costs; 75% or more of the costs are due to enhancements.

Prerequisite knowledge for Complete understanding and learning of Topic:

(Max. Four important topics)

- Types of Software maintenance
- Software process and activities
- Process models
- Reverse Re-Engineering

Detailed content of the Lecture:

Factors that mandate change:

- Bug Fixes
- Change in operating environment
- Change in government policies and regulations
- Change in business Procedures
- Changes to prevent future problems

LEHMAN'S Law of System Evolution

- Law of Continuing Change
- Law of increasing entropy or complexity
- Law of Self Regulation
- Law of Conservation
- Lawof continuinggrowth

Types of Software Maintenance

- Corrective maintenance
- Adaptive maintenance
- Perfectivemaintenance
- Emergency maintenance

Software Maintenance Process and Activities

1. Program understanding
2. Change identification and Analysis

3. Configuration Change Control
4. Change implementation, testing and delivery.

Maintenance Process Models

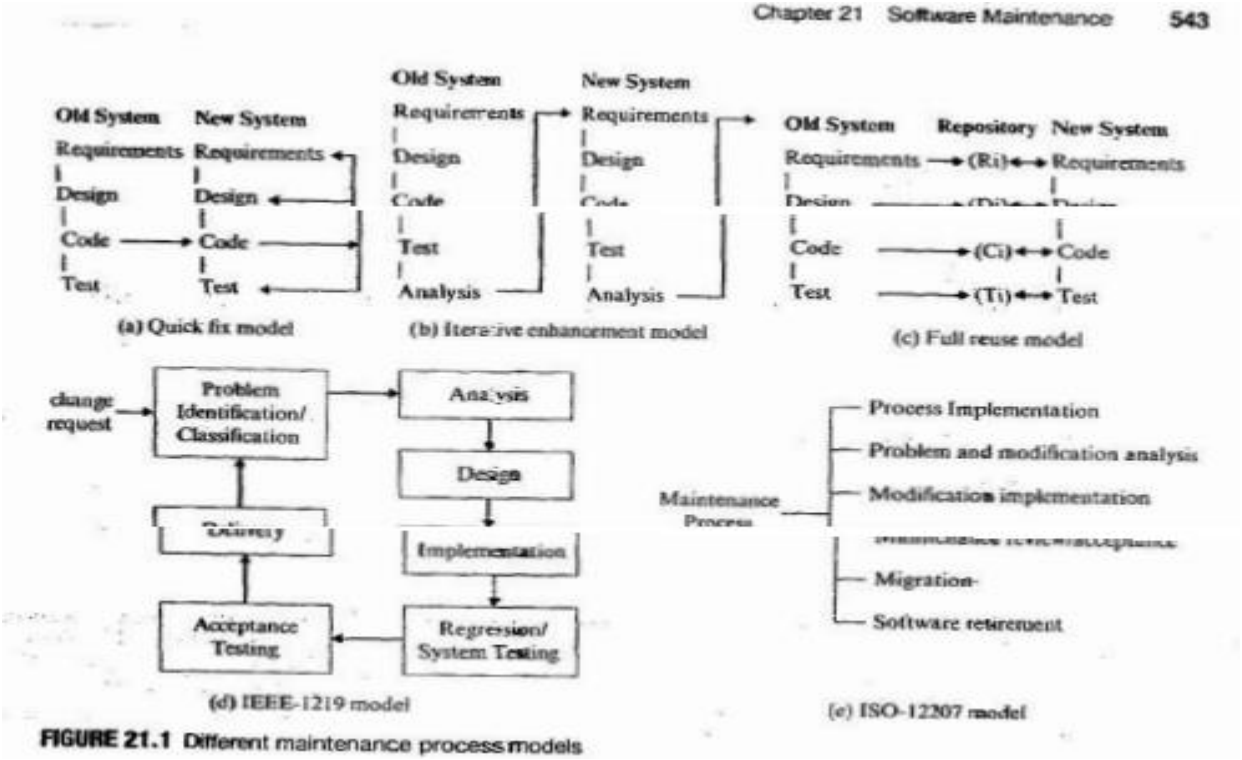


FIGURE 21.1 Different maintenance process models

Program Understanding

To change a software system, the software engineer needs to understand the program. This is commonly too referred as program understanding or program comprehension.

It involves a process mat extracts the design and specification artifacts from the code and represents them in a mental model.

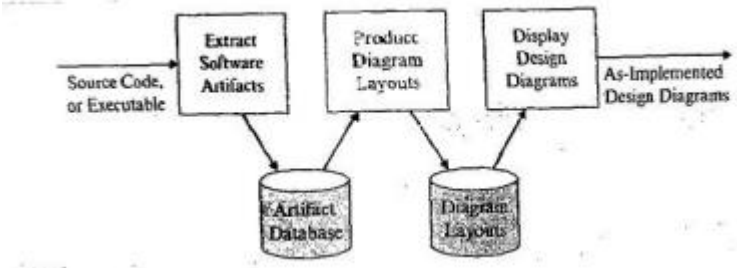
Change Identification and Analysis

1. Assess the change irnpact that is, which other components will be affected by the changes made to a given component.
2. Estimate the costs and time required to implement the changes and test the result.
3. Identify risks and define resolution measures.

Configuration Change Control

- Preparing an engineering change proposal.
- Evaluating the engineering change proposal.

REVERSE-ENGINEERING



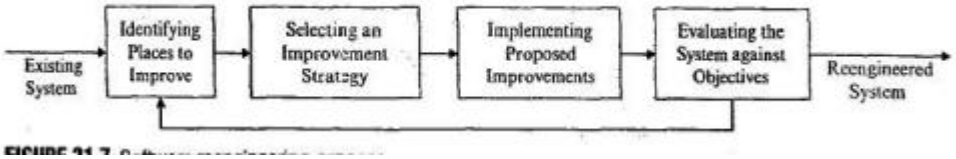
Usefulness of Reverse engineering

- Program Understanding
- Formal Analysis
- Test case generation

Software Re-engineering

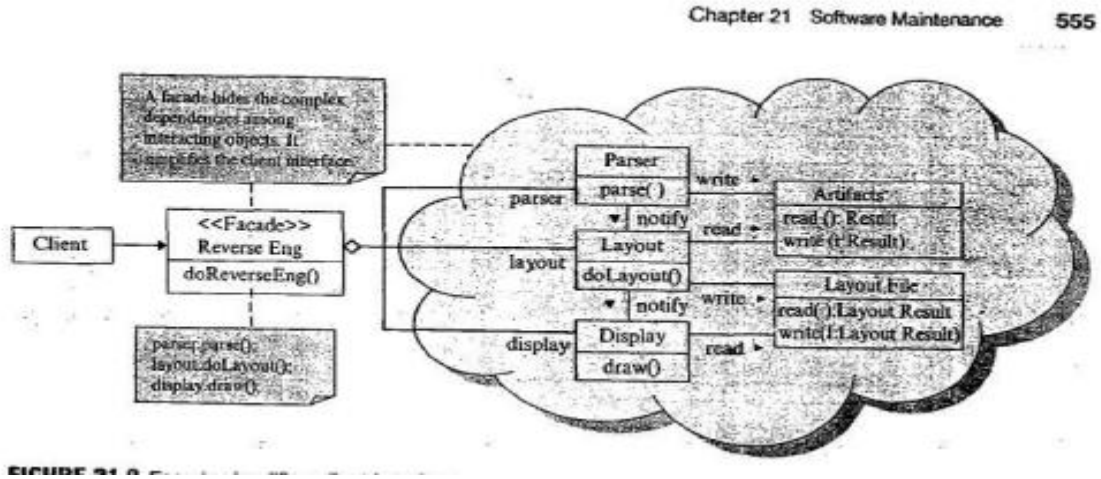
Objectives of Re engineering

Improving the software architecture
Reducing the complexity of software
Improving the ability to adapt change
Improving the maintainability



Patterns for Software Maintenance

- Simplifying Client interface with façade
- Simplifying component interaction with mediator



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

Can be added as link

https://www.geeksforgeeks.org/software-engineering-verification-and-validation/
https://www.youtube.com/watch?v=qxitgylm1EU

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

David Kung Object-Oriented Software Engineering:An Agile Unified Methodology McGraw-Hill Education 2013Page No:483-490

Journals:

https://www.researchgate.net/journal/1532-0618
https://www.scimagojr.com/journalsearch.php?q=89465&tip=sid&clean=0

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : Object Oriented Software Engineering-16CSD04

Course Faculty :

Unit : V **Date of Lecture:**

Topic of Lecture: Software Configuration Management

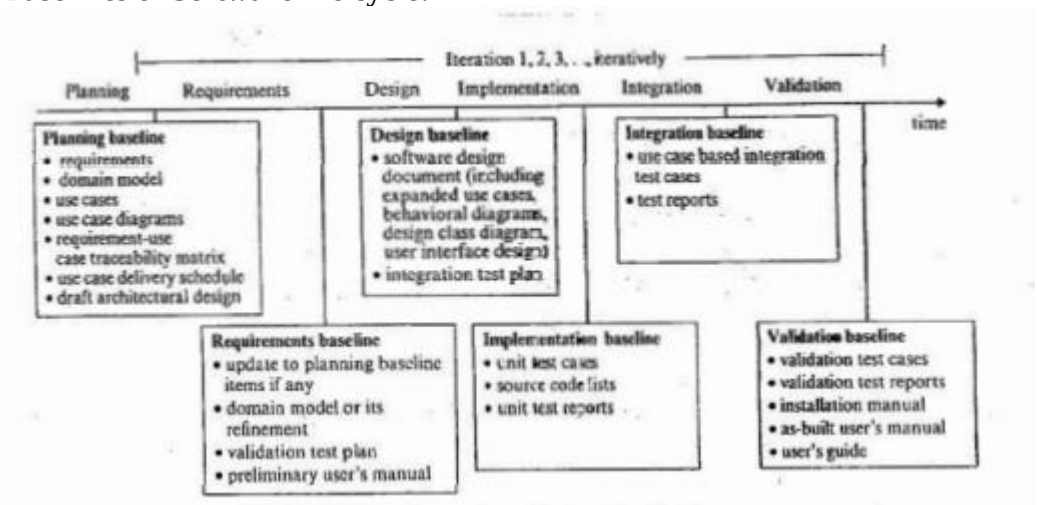
Introduction : (Maximum 5 sentences) :

- Software configuration management (SCM) is baseline and configuration item management.
- SCM consists of configuration item identification, configuration change control, configuration auditing, and configuration status accounting.
- During the software life cycle, numerous documents are produced.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- Life Cycle
- SCM Functions
- SCM Tools

Detailed content of the Lecture:
Baselines of Software life cycle:



SOFTWARE CONFIGURATION MANAGEMENT FUNCTIONS

- Software Configuration Identification
- Configuration Change Control
- Software Configuration Auditing
- Software Configuration status accounting

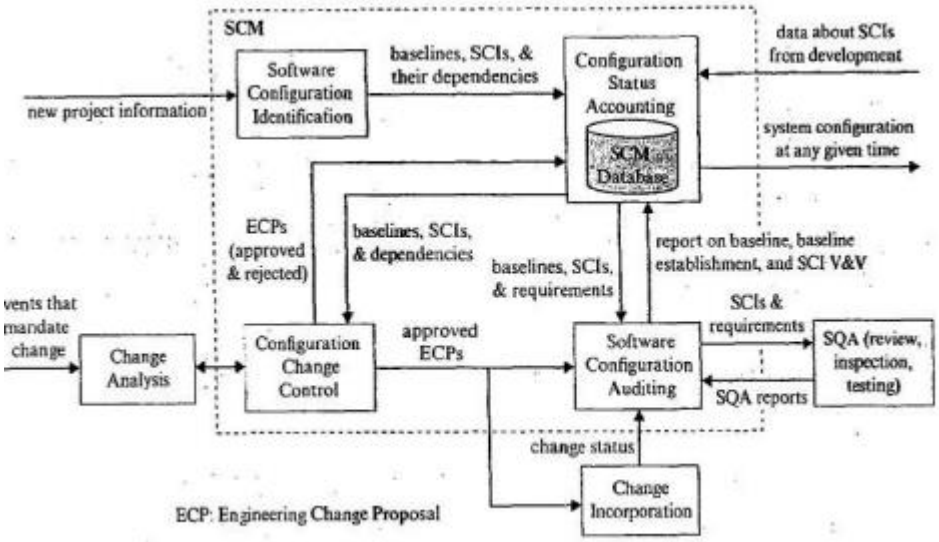
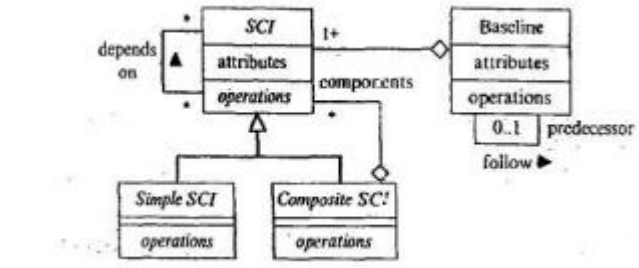


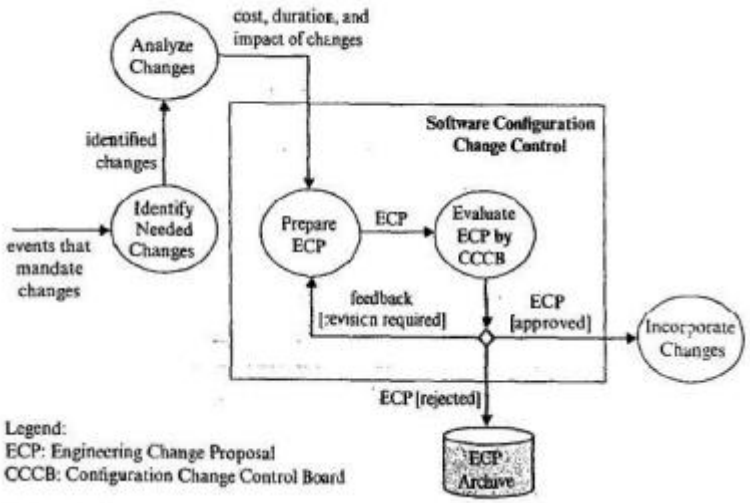
FIGURE 22.2 Software configuration management and process

Software Configuration Identification



- ID Number
- Name
- Document Type
- Document File
- Author
- Version Number
- Update history
- Description

Software Configuration Change Control



Software Configuration Auditing

- Defining mechanisms for establishing and formally establishing a baseline.
- Configuration item verification
- Configuration item validation
- Ensuring that changes specified

CONFIGURATION MANAGEMENT IN AN AGILE PRGJECT

Agile projects welcome change and need to respond to changes rapidly.
However, conventional configuration management involves a rigorous and often lengthy

change control process.

SOFTWARE CONFIGURATION MANAGEMENT TOOLS

- Version Control
- Workspace Management
- Concurrency Control
- System Build
- Support to SCM Process

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

https://www.tutorialspoint.com/software_engineering/software_project_management.htm
https://www.youtube.com/watch?v=AaHaLjuzUm8

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

David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 Page No: 562-570

Journals

https://ieeexplore.ieee.org/document/6772879
https://www.researchgate.net/publication/220773173

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **V** **Date of Lecture:**

Topic of Lecture: Project Organization

Introduction : (Maximum 5 sentences) :

- Managing a software project must address a number .of issues relating to project organization.
- That is, how the reams are formed, how the teams and team members work together to carry out the life-cycle activities.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- Effort estimation method
- COCOMO II Model
- The Delphi estimation method

Detailed content of the Lecture:

Project Format

The project format is concerned with how the life-cycle activities are assigned to the project teams. Three project formats have been used in practice:

- project-based format
- function-based format
- hybrid format.

Team Structure

The team structure is concerned with the organization of the project teams, that is, assigning roles and responsibilities to the team members.

- Egoless Team Structure
- Chief programmer team structure
- Hierarchal team Structure

EFFORT ESTIMATION METHODS

The Function Point Method

The function point (FP) of a system is a product of the gross function point (GFP) and the processing complexity adjustment (PCA).

$$GFP = \sum_{i=1}^{5} (Count_i \times Complexity_i)$$

COCOMO Model II

The Application Composition Model

The application composition model is used during the early stages of the life cycle to estimate effort required to build a prototype. It is also used for projects that construct systems from commercial off-the-shelf (COTS) software components.

The Early-Design Model

The early design model is used in tile early stages of a software project when very little about the software size and the target environment is known. The basic formula for effort calculation is:

$$\text{Effort } PM = a \times size^{b} \times \prod_{i=1}^{n} EM_i$$

Estimate Software Size

The software size in thousand source lines of code (KSLOC) can be estimated in two different ways: direct estimation or using function points.

- External inputs
- External Outputs
- Internal Logical files
- External interface files
- External Queries

The Delphi Estimation Method

The Delphi estimation method relies on a group of experts to produce the estimation. It has the following

Step: 1. Form a group of experts or experienced developers.

Step: 2. Present an overview of the system and its major components to the group.

Agile Estimation

Agile processes welcome change. Therefore, agile estimation means that the effort estimates can and must change to match the reality. Agile processes believe that good enough is enough.

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

https://www.geeksforgeeks.org/layers-of-osi-model/

https://www.youtube.com/watch?v=EzLMMsRR6Js

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

David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 PAGE NO:577-579

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **V** **Date of Lecture:**

Topic of Lecture: Effort Estimation Methods

Introduction : (Maximum 5 sentences) :

- Managing a software project must address a number .of issues relating to project organization.
- That is, how the reams are formed, how the teams and team members work together to carry out the life-cycle activities.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- Effort estimation method
- COCOMO II Model
- The Delphi estimation method

Detailed content of the Lecture:

Project Format

The project format is concerned with how the life-cycle activities are assigned to the project teams. Three project formats have been used in practice:

- project-based format
- function-based format
- hybrid format.

Team Structure

The team structure is concerned with the organization of the project teams, that is, assigning roles and responsibilities to the team members.

- Egoless Team Structure
- Chief programmer team structure
- Hierarchal team Structure

EFFORT ESTIMATION METHODS

The Function Point Method

The function point (FP) of a system is a product of the gross function point (GFP) and the processing complexity adjustment (PCA).

$$GFP = \sum_{i=1}^{5} (Count_i \times Complexity_i)$$

COCOMO Model II

The Application Composition Model

The application composition model is used during the early stages of the life cycle to estimate effort required to build a prototype. It is also used for projects that construct systems from commercial off-the-shelf (COTS) software components.

The Early-Design Model

The early design model is used in tile early stages of a software project when very little about the software size and the target environment is known. The basic formula for effort calculation is:

$$\text{Effort } PM = a \times size^{b} \times \prod_{i=1}^{n} EM_i$$

Estimate Software Size

The software size in thousand source lines of code (KSLOC) can be estimated in two different ways: direct estimation or using function points.

- External inputs
- External Outputs
- Internal Logical files
- External interface files
- External Queries

The Delphi Estimation Method

The Delphi estimation method relies on a group of experts to produce the estimation. It has the following

Step: 1. Form a group of experts or experienced developers.

Step: 2. Present an overview of the system and its major components to the group.

Agile Estimation

Agile processes welcome change. Therefore, agile estimation means that the effort estimates can and must change to match the reality. Agile processes believe that good enough is enough.

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

https://www.geeksforgeeks.org/layers-of-osi-model/

https://www.youtube.com/watch?v=EzLMMsRR6Js

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

David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 PAGE NO:577-579

Course Faculty

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-

LECTURE HANDOUTS

CSE

II/IV

Course Name with Code : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **V** **Date of Lecture:**

Topic of Lecture: Planning and Scheduling

Introduction : (Maximum 5 sentences) :

- Project planning and scheduling are concerned with the scheduling of development activities and allocation of resources to the development activities.
- Project planning and scheduling are critical to the success of a project because poor planning may result in schedule slippage, cost overrun, poor software quality, and/or high maintenance costs.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- PERT Chart
- GANTT Chart

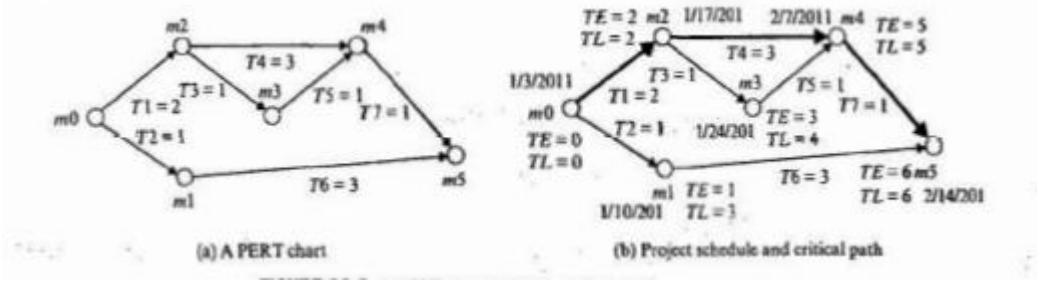
Detailed content of the Lecture:

PERT Chart

The program evaluation and review technique (PERT) chat is widely used for project scheduling.

A PERT chart is an edge-weighted directed graph or digraph G = (M, T, D), where M is the set of vertexes representing project milestones, D a set of labels denoting tasks durations, and T 0; If x M x D the set of directed edges representing the project tasks and their durations.

The d graph satisfies the following conditions:
• It does not have parallel edges.
• It does not have cycles, that is, the digraph is acyclic
• It has exactly one source and one sink. The source represents the start of the project and the sink the completion of the project.

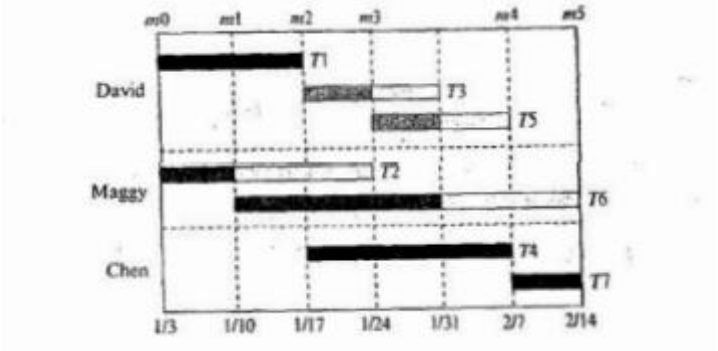


(a) A PERT chart (b) Project schedule and critical path

Gantt Chart and Staff Allocation

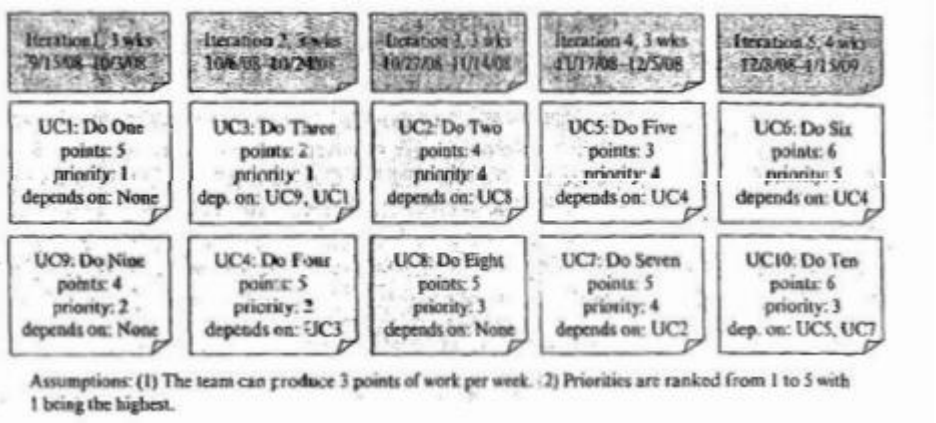
The PERT chart is a useful tool for computing the earliest start time and latest completion time for each of the milestones as well as the earliest completion time of the project. But a PERT chart is not intuitive in showing the progression of the tasks and the amount of time available for each of the tasks.

The Gantt chart is a better tool for highlighting such information.



Agile Planning:

It is easy to ensure that high-priority use cases are developed and deployed earl y. It is easy to ensure that die dependencies between the use cases are satisfied.



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

https://www.geeksforgeeks.org/layers-of-osi-model/
https://www.youtube.com/watch?v=EzLMMsRR6Js

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

David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 Page No:591-594

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **V** **Date of Lecture:**

Topic of Lecture: Risk Management

Introduction : (Maximum 5 sentences) :

- Many contingencies could negatively impact a software project. Sometimes, the consequence of such an event is unbearable.
- The National Health System project and the Textile Process Control project discussed at the beginning of this chapter could have been avoided if proper risk analysis had been performed. Such contingent events are commonly referred to as risks.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- Risk Identification
- Risk Analysis
- Planning
- Monitoring

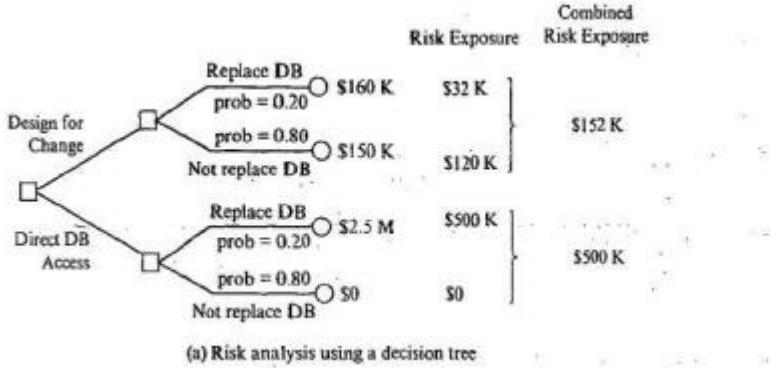
Detailed content of the Lecture:

Risk Identification

Risks can be classified into universal project risks and project-specific risks. The universal project risks are risks that can occur to all projects while project-specific risks are risks that can occur only to a particular project.

Risk Analysis and Prioritizing

- Risk analysis is concerned with the determination of the extent of damage of each risk, options to deal with the risk and costs to implement the options.
- The analysis is aimed to determine which option to take, the cost to implement that option, and the extent of damage with that option.
- Risk analysis involves several basic concepts, that is, the loss probability loss magnitude, and risk exposure.



(a) Risk analysis using a decision tree

Risk Management Planning

- Risk analysis and prioritizing identify a list of risk items, compute their combined risk exposures, and rank them with priorities.
- The next step of risk management is producing a risk management plan to be carried out during the development process.
- The first step of risk management planning is developing strategies to address the risk items. The risk management techniques shown in the right-most column.

Risk Resolution and Monitoring

- Risk resolution is the implementation and execution of the risk reduction techniques specified and scheduled in the risk management plan.
- Risk monitoring ensures that the risk reduction strategies are implemented and executed according to schedule.
- It is aimed to ensure that the risk management process is a closed-loop process and progresses on track. Rather than monitoring an risk items, it is more effective to focus on the top-N risk items of the project, where N should be limited to 10, and depends on the project size, nature, and progress status.
- The status of the top-N risk items is updated to reflect changes of their rankings from the last review, number of months on the list, and risk-resolution status.

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

https://www.geeksforgeeks.org/layers-of-osi-model/
https://www.youtube.com/watch?v=EzLMMsRR6Js

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

David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 Page No: 595-599

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : **Object Oriented Software Engineering-16CSD04**

CourseFaculty :

Unit : **V** **Date of Lecture:**

Topic of Lecture: Process Improvement

Introduction : (Maximum 5 sentences) :

- A software process defines a series of activities for constructing a software system.
- The execution of a software process has to be monitored and data about various aspects of the process including productivity, quality, costs, and time to market should be collected.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- CMMI Model
- Software process

Detailed content of the Lecture:

The CMMI was originally developed by the Software Engineering Institute (SEI) to assist the U.S. Department of Defense (DDD) to assess the performance of the defense contractors.

During the years, the CMMI has expanded its acceptance beyond the defense industry; currently, it is widely used by many software development organizations.

The CMMI model has a number of merits:

• It reflects the actual process improvement practices. For example, studies show that the delivered defect densities or delivered defects per 1,OOO.tines of code for CMMI level I to level S are 7.5, 6_24, 4.73, 2_28, and l.05;respectively.

• For each level, it clearly defines the improvement goals and progress measures.

• The five maturity levels define a logical roadmap toward an optimizing process. The improvement from one level to the next higher level can usually be achieved in two years.

• The recommendation provides improvement priorities.

To improve the software process, an organization performs the following steps, which may serve as a self-study:

1. Evaluate the current process to gain an understanding of its status, that is, what is the maturity level of the current process.

2. Develop a vision for the desired process at the next higher maturity level, guided by the key process areas.

3. Define a plan of prioritized actions for improvement. 4. Implement the action plan. 5. Repeat the above steps to move to the next high level.

Level	Name	Characteristics	Key Process Areas
1	Initial Level	• An ad hoc/chaotic process • No project management mechanism, no cost estimation, no project plans • Tools are not well integrated • Change control is lax • Senior management does not understand key issues	None
2	Repeatable Level	• An intuitive process that depends on individuals • Established basic project controls • Strength in similar work but facing significant risks with new challenges • Lacks an orderly framework for improvement	• Requirements management • Software project management • Subcontractor management • Software quality assurance • Software configuration management
3	Defined Level	• An organization-wide qualitative process is defined and implemented • A process group to improve process	• Organization process definition and focus • Training program • Integrated development and management • Software product engineering • Intergroup coordination • Peer reviews
4	Managed Level	A quantitative process with a process database, and a minimum set of quality and productivity measures	• Quantitative process management • Software quality management
5	Optimizing Level	An ever-improving process that is supported by • automatic data collection • using data to identify weaknesses • rigorous defect-cause analysis & defect prevention • numeric evidence to justify technology use • improvement feedback into process	• Defect prevention • Technology change management • Process change management

Video Content / Details of website for further learning (if any):
https://www.geeksforgeeks.org/layers-of-osi-model/
https://www.youtube.com/watch?v=EzLMMsRR6Js

Important Books/Journals for further learning including the page nos.:
David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 Page No:599-600

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **V** **Date of Lecture:**

Topic of Lecture: Software Security in the Life Cycle

Introduction : (Maximum 5 sentences) :

- Software security is a proactive, rather than reactive, approach to constructing secure software.
- Consideration of software security should begin in the requirements phase and continue throughout the life cycle.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- Software Security
- Security requirements
- Design principles

Detailed content of the Lecture:

- Basic concepts of software security
- Importance of software security.
- Security attacks and defenses.
- Security requirements.
- Secure software design principles.
- Security patterns.
- Life-cycle activities for building secure software.

SOFTWARE SECURITY

- Modeling and analysis for security.
- Design for security.
- Secure coding
- Test for security
-

SECURITY REQUIREMENTS

- Identification requirements
- Authentication requirements
- Immunity requirements
- Integrity requirements
- Privacy requirements

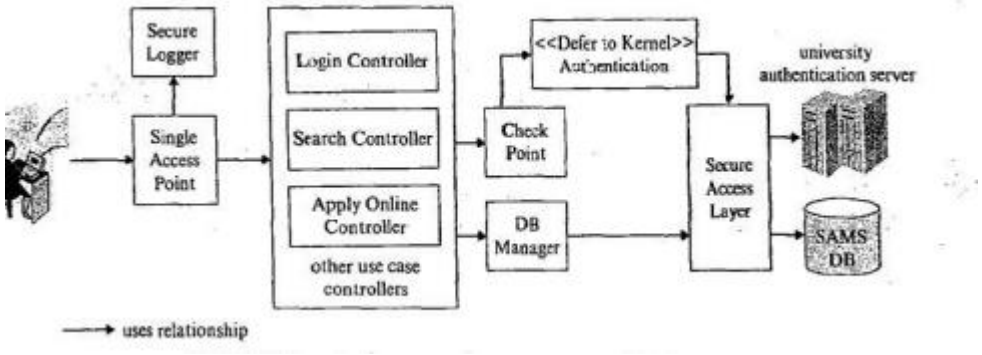
SECURE SOFTWARE DESIGN PRINCIPLES

- Secure the weakest link.
- Practice defense in depth.
- Fail securely.
- Least privilege.
- Compartmentalize.
- Keep it simple and stupid.

SECURE SOFTWARE DESIGN PATTERNS

These software design patterns help software developers produce quality software while improving teamwork, communication and productivity.

Patterns are proven design solutions to commonly encountered design problems.



RISK ANALYSIS WITH AN ATTACK TREE

The architectural risk analysis and misuse cases can benefit from the use of attack trees, which are derived from the fault tree analysis technique.

There are two types of node:

- AND-node
- OR-node.

An AND-node means that the problem is solved if all of its child problems are solved.

An OR-node, which is the default, means the problem is solved if one of its child problems is solved.

SOFTWARE SECURITY IN THE LIFE CYCLE

Security in the Planning Phase

Deriving Security Requirements

Software security is aimed at building software systems that possess the ability to thwart security attacks and recover from successful attacks.

Identifying Misuse Cases

Building secure systems must ensure that the security requirements are complete and adequate, and the security mechanisms are properly implemented.

If these conditions are not met, then attackers could exploit the flaws to launch attacks.

Producing a Secure Architecture

1. Produce an architectural design that satisfies the security requirements and accounts for misuse cases.
2. Evaluate the architectural design to identify significant security risks.
3. Modify the architectural design to remove or mitigate the significant security risks.
4. Repeat the last two steps until an acceptable risk level is achieved.

Security in the Iterative Phase

Security in Requirements Change

Requirements change is a common practice .n today's software development. Requirements can change as often as every day or every week, especially at the beginning of an agile project.

Security in Behavioral Design

The behavioral design activities include design of expanded use cases, sequence diagrams, state diagrams, activity diagrams, and derivation of a design class diagram.

Security in implementation, Testing, and Deployment

1. Guiding implementation with secure software design principles. Many secure software design principles are applicable to implementation.
2. Applying implementation-level security patterns.
3. Practice secures programming principles and practices.
4. Testing for security.

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

https://resources.infosecinstitute.com/intro-secure-software-development-life-cycle/
https://dzone.com/articles/how-to-approach-security-development-lifecycle-sdl

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

David Kung, Object-Oriented Software Engineering: An Agile Unified Methodology, McGraw-Hill Education,2013 PAGE NO:614-623

Course Faculty

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

LECTURE HANDOUTS

L-

CSE

II/IV

Course Name with Code : **Object Oriented Software Engineering-16CSD04**

Course Faculty :

Unit : **V** **Date of Lecture:**

Topic of Lecture: Applying Agile Principles- Software Tools

Introduction : (Maximum 5 sentences) :

- Conventional approaches treat maintenance as a post-development activity.
- For an agile project, maintenance begins with the delivery of the first increment or release development is maintenance and maintenance is development.

Prerequisite knowledge for Complete understanding and learning of Topic:
(Max. Four important topics)

- Software Security

Detailed content of the Lecture:

- The maintenance process models also have the requirements, design, implementation, and testing phases as in the development process.
- This implies that the agile principles applicable to the development phases are also applicable to the phases of the maintenance process.
- Therefore, the following only presents principles that are specific to maintenance.

GUlDELINE: Good enough is enough.

- Improving the structure of the software system is important because it reduces the maintenance costs. However.
- Perfective maintenance is not aimed at obtaining the perfect architecture. In fact, the perfect or optimal architecture does not exist. A good enough architecture is good enough

TOOLS SUPPORT fOR SOFTWARE MAINTENANCE

- Many software maintenance activities are tedious and time consuming. Moreover, software maintenance needs to coordinate the changes to ensure consistency.
- The resulting software system needs to be retested to ensure that it satisfies the requirements and constraints.
- The use of software tools can significantly reduce the time and effort.

The following are some of the tools that are useful for software maintenance:

Reverse-engineering tools are useful for design and specification recovery. They aid program comprehension and identification of places that need improvement.

- These tools are extremely valuable when the design documentation is missing, outdated, or inadequate.

Metrics calculation tools compute and display quantitative measurements of a software system.

- They help in identifying and highlighting places that need improvement. For example, classes that consist of thousands of lines of code are difficult to maintain and are more likely to be error prone.
- Classes that have an excessive number of functions may be assigned too many responsibilities. Methods with a high complexity are candidates for improvement.

Performance measurement tools such as software profilers can display execution times, invocation Frequencies, and memory usage of various components of a software system.

- They are useful for identifying performance bottlenecks and memory-intensive components, Software reengineering may be needed to mitigate these problems.

Static analysis tools are useful for detecting violation of coding standards, incorrect use of types, existence of certain bogs and anomalies, and security vulnerabilities.

Change impact analysis tools are useful for assessing the scope of impact of proposed improvements.

- The change impact analysis results are the basis for the estimation of the effort required to perform the proposed improvements.

Effort estimation tools are useful for calculating the required time, effort, and costs to implement the proposed improvements.

Configuration management tools such as Coocurrent Versions System (CVS) and Subversion are useful for coordinating tile changes to maintain the consistency of the software being reengineered.

Regression testing tools are useful for rerunning the test cases to ensure that the system satisfies the requirements and reengineering does not introduce new errors.

- Some of the tools can analyze the software and select a subset of test cases to rerun. This reduces the regression testing time and effort.

Pattern	Type	Example Applications/Benefits
Abstract factory	AEP	• Abstract factory can create objects that are environment or platform dependent. Thus, it can be applied to adaptive and perfective maintenance. • Abstract factory can be used to add new product families. Thus, it is applicable to enhancement maintenance.
Adapter	ACEP	• ACEP types of maintenance may reuse an existing component. Adapter can be used to adapt the existing interface. • It is useful for the full reuse process.
Bridge	AEP	• Bridge allows the interface and implementation to change independently. This makes the software easily adapt to changing environment. • New functionality can be added easily; and hence, it supports enhancement maintenance. • Applying bridge improves the ability of the software in many aspects including ease to maintain and adaptability to changes in requirements and environment.
Builder	AEP	• Builder can be used to enhance or improve the software to support new processes or new process steps. • Concrete supervisors and builders can adapt the software to changing environment. • Useful for maintaining enterprise resource planning (ERP) software.
Chain of responsibility, Controller, Observer	EP	• These patterns decouple event sources and handlers. Thus, it is easy to add handlers or sources to support enhancement and perfective maintenance. • Decoupling implies reduction of change impact; and hence, they facilitate software maintenance.
Command	EP	• Command is a special case of polymorphism. Therefore, it can be used to eliminate some of the conditional statements. Complexity is reduced. • It reduces the size of a class by delegating its functions to command objects. • It makes the software easy to add new type of command.
Composite	EP	• Composite simplifies the client's processing. • It improves the ability of the software to represent complex structures. • It makes the software easy to add new primitives or composites.
Decorator, Visitor	EP	• These patterns can add functionality to existing objects dynamically. New decorator or visitor can be added easily; and hence, they support enhancement maintenance. • They can remove functionality from an existing object and assign it to a decorator or visitor. This improves the cohesion of the object as well as reducing the use of conditional statements.
Facade, Mediator	P	• Facade simplifies the client interface and decouples it from the components. Mediator simplifies the interaction among the components. • They facilitate maintenance because (1) the software is easy to understand, and (2) decoupling reduces the change impact of the components. • They facilitate reuse. Facade makes the client easy to reuse the components. Mediator facilitates reuse of any of the components.
Factory method, Template method	AEP	• The concrete subclasses may implement environment or platform-dependent behavior; and hence, it can be used for adaptive maintenance. • Subclasses can be added easily; and hence, it facilitates enhancements. • These two patterns make the code easy to understand, modify, and reuse; and hence, it improves software maintainability.
Flyweight, Singleton, Virtual proxy, Smart reference proxy	P	• These patterns improve the efficiency or performance of the software. • Flyweight and singleton reduce the number of objects created. • Virtual proxy delays the creation of objects that are time consuming to create or memory intensive. • Smart reference proxy keeps track of object use; and hence, it improves performance and efficiency.

FIGURE 21.12 Patterns useful in the maintenance phase

Interpreter	EP	• Interpreter allows business rules to be updated dynamically. • It is easy to add or modify rules; and hence, it supports enhancement and perfective maintenance.
Iterator	P	• Iterator hides the implementation of the collection and makes maintenance easier.
Protection proxy	CE	• Protection proxy controls access to an object. • It can be used to add protection to correct security and concurrent access problems. Likewise, such protection can be added as enhancement to existing software.
Prototype	EP	• Prototype reduces the number of classes and hence maintenance is easier. • Prototype supports dynamically loaded classes. This can be explored to support dynamic addition of functionality.
State	EP	• State simplifies the design and implementation of state behavior. It makes the software easy to understand, test, and maintain. • New states and new events can be added easily; and hence, it supports enhancement maintenance.
Strategy	EP	• Strategy encapsulates algorithms as objects. Thus, it is easy to add new algorithms as enhancement maintenance. • It reduces the use of conditional statements to select the strategy; and hence, it makes the software easy to maintain.

Note: Type=maintenance type A=Adaptive, C=Corrective, E=Enhancement, P=Perfective

Video Content / Details of website for further learning (if any):
https://www.crossware.co.nz/blog/top-5-lotus-notes-software-tools/
https://www.slideshare.net/ravindravekariya/software-tools-38364252

Important Books/Journals for further learning including the page nos.:
David Kung Object-Oriented Software Engineering: An Agile Unified Methodology McGraw-Hill Education 2013. Page no:627-628

Course Faculty

Verified by HOD