BIT 2321 SOFTWARE ENGINEERING

Prerequisite: *ICS 2302 SOFTWARE ENGINEERING 1*

Purpose: this course aims to polish a student for software development in the radical and distributed computing environments. Provide the student with the knowledge of building three tier and enterprise applications. It introduces current principles in software development and collaboration and standards of middleware.

COURSE OUTLINE

[LESSON ONE 6](#_Toc396042680)

[Introduction to software engineering II 6](#_Toc396042681)

[1.1 What is software engineering 6](#_Toc396042682)

[1.1.1 The software engineering process model has the following steps 6](#_Toc396042683)

[1.2 Application of software techniques in today’s changing economy. 6](#_Toc396042684)

[LESSON TWO 7](#_Toc396042685)

[Software design patterns and collaborative development 7](#_Toc396042686)

[2.1 Introduction to software design patterns that can be used in software engineering 7](#_Toc396042687)

[2.2 Java Design Pattern’s. 7](#_Toc396042688)

[2.3 Introduction to the java framework (MVC, struts, spring, JSF) 10](#_Toc396042689)

[2.3.1 Benefits of the MVC model 10](#_Toc396042690)

[2.4 How to implement selected design pattern in java 12](#_Toc396042691)

[LESSON THREE 15](#_Toc396042692)

[Component based technologies used in software engineering 15](#_Toc396042693)

[3.1 Introduction to component based engineering and technologies. 15](#_Toc396042694)

[3.2 Importance of component based engineering in software development 15](#_Toc396042695)

[3.3 Application of Component based technologies in software development 15](#_Toc396042696)

[3.4 COM 16](#_Toc396042697)

[3.5 EJB 17](#_Toc396042698)

[3.6 CORBA 18](#_Toc396042699)

[3.7 Component collaborative development process 19](#_Toc396042700)

[LESSON FOUR 21](#_Toc396042701)

[Middle ware enterprise development and service oriented programming 21](#_Toc396042702)

[4.1 Introduction to middle ware enterprise development. 21](#_Toc396042703)

[4.2 Importance of middle ware development in enterprise system development 22](#_Toc396042704)

[4.3 Model Integration Computing 22](#_Toc396042705)

[4.4 Integrating MIC with component middleware 23](#_Toc396042706)

[4.5 Service oriented programming and its application in software engineering. 27](#_Toc396042707)

[4.5.1 Service oriented programming elements 27](#_Toc396042708)

[4.5.2 Aspects of service oriented programming 28](#_Toc396042709)

[4.5.3 Benefits of Service Oriented Programming 28](#_Toc396042710)

[LESSON FIVE 29](#_Toc396042711)

[Software validation and verification 29](#_Toc396042712)

[5.1 Introduction to software validation and verification 29](#_Toc396042713)

[5.2 The difference between validation and verification of software 29](#_Toc396042714)

[5.2.1 Where is each applicable in the development process of the software 29](#_Toc396042715)

[5.3 Introduction to software testing 29](#_Toc396042716)

[5.4 Testing procedures and strategies that can be used to test the software 30](#_Toc396042717)

[5.5 Different stages of testing for the software 30](#_Toc396042718)

[5.6 Importance of validating and verifying software. 31](#_Toc396042719)

[LESSON SIX 32](#_Toc396042720)

[Computer software Project management 32](#_Toc396042721)

[6.1 Introduction computer software project management 32](#_Toc396042722)

[6.2 Activities in Software project management 32](#_Toc396042723)

[6.3 Methodologies in software project management 33](#_Toc396042724)

[6.3.1 Rational Unified Process (RUP) 33](#_Toc396042725)

[6.3.2 SSADM 34](#_Toc396042726)

[6.4 Techniques in software project management 34](#_Toc396042727)

[6.4.1 Project Management body of Knowledge (PMBOK) 34](#_Toc396042728)

[6.4.2 COCOMO 35](#_Toc396042729)

[6.4.3 Milestone Trend Analysis 35](#_Toc396042730)

[6.4.4 Earned value management (EV) 36](#_Toc396042731)

[LESSON SEVEN 38](#_Toc396042732)

[Object Oriented Software development 38](#_Toc396042733)

[7.1 Introduction to object oriented software development 38](#_Toc396042734)

[7.2 Use of UML diagrams in O.O software development technique to model/design software. 39](#_Toc396042735)

[7.3 TERMS AND CONCEIPTS 39](#_Toc396042736)

[7.3.1 Class and object. 39](#_Toc396042737)

[7.3.2 Data Abstraction and Encapsulation 40](#_Toc396042738)

[7.3.4 Access modifiers/rules of encapsulation 41](#_Toc396042739)

[7.3.5 Polymorphism and dynamic binding. 42](#_Toc396042740)

[7.4 BENEFITS OF OOP 47](#_Toc396042741)

[7.5 APPLICATIONS OF OOP 47](#_Toc396042742)

[LESSON EIGHT 52](#_Toc396042743)

[Computer software Cost estimation. 52](#_Toc396042744)

[8.1: Introduction to software cost estimation 52](#_Toc396042745)

[8.1.1 Accurate cost estimation is important because: 52](#_Toc396042746)

[8.2: Methods used to estimate software cost 52](#_Toc396042747)

[8.2.1 Algorithmic methods 53](#_Toc396042748)

[8.2.2 Non algorithmic methods 53](#_Toc396042749)

[8.3 COCOMO I 53](#_Toc396042750)

[8.4: Function point 54](#_Toc396042751)

[8.5: Software Line of Code 55](#_Toc396042752)

[LESSON NINE 56](#_Toc396042753)

[Quality management and software maintenance 56](#_Toc396042754)

[9.1 Introduction to software maintenance 56](#_Toc396042755)

[9.1.1. Software maintenance activities 56](#_Toc396042756)

[9.1.2. Techniques for maintenance 58](#_Toc396042757)

[9.3 Quality management and importance in software engineering. 58](#_Toc396042758)

[9.3.1 Classification of Quality requirements 59](#_Toc396042759)

[9.3.2 Software Quality Architecture 60](#_Toc396042760)

[9.4 Configuration management and its importance in software engineering 61](#_Toc396042761)

[LESSON TEN 62](#_Toc396042762)

[Software prototype and formal specification 62](#_Toc396042763)

[10.1 Introduction to software prototyping. 62](#_Toc396042764)

[10.1.1 Software prototyping strategies 63](#_Toc396042765)

[10.2 Online rapid prototyping techniques 64](#_Toc396042766)

[10.3 Offline rapid prototyping techniques 64](#_Toc396042767)

# LESSON ONE

# Introduction to software engineering II

Learning outcomes:

Upon completion of this lesson, the learner should be able to:

- Recap the definition of software engineering as learnt in software engineering one

- Appreciate the need for software engineering in developing high end and quality enterprise software’s.

## 1.1 What is software engineering

Before even we define the term software engineering, we should ask ourselves to define the term software. Software is an aspect in computing that deals with providing an abstraction that helps the user interact with hardware on the machine. Therefore software helps us users accomplish a certain task. Software can be tailor made or general purpose software.

Software engineering is therefore the science and art of building significant software systems that are on time, within the budget, performing the correct operation and within the stipulated user requirements.

* + 1. The software engineering process model has the following steps:
* **Specification**: Set out the requirements and constraints on the system.
* **Design**: Produce a model of the system.
* **Manufacture**: Build the system.
* **Test**: Check the system meets the required specifications.
* **Install**: Deliver the system to the customer and ensure it is operational.
* **Maintain**: Repair faults in the system as they are discovered.

## 1.2 Application of software techniques in today’s changing economy.

Software engineering is mostly used in developing robust and agile applications and software’s in our ever changing technological world. From military applications to basic home applications, software engineers apply the methods and principles from the software engineering practices that have evolved over the years.

## 1.3 Sample Questions

a) Define the term software engineering; in addition explain its importance in the software development process.

# LESSON TWO

# Software design patterns and collaborative development

**Learning outcomes**

Upon completion of the lesson, the learner should be able to:

* Be able to describe, recognize, apply and implement selected design patterns in java.
* Be able to implement these designs in software project
* To explain the link between a selected design pattern and the design methodology of choice in use.

## Introduction to software design patterns that can be used in software engineering

Patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context. Patterns are used in software engineering to generalize a solution to a particular design problem which shall be a blue print in essence for the software.

The following are elements of a design pattern:

1. Pattern name: useful vocabulary that is used to describe the pattern.
2. Problem applicability and area of applicability: describes the area where the pattern can be applied and the problem it can solve.
3. Solution: the solution to the problem to which it was designed for in that problem domain.
4. Consequences of applying that design pattern: it should specify the course of the whole process, the pros and cons of applying that design pattern.

## Java Design Pattern’s.

Design patterns represent the best practices used by experienced object-oriented software developers. Design patterns are solutions to general problems that software developers faced during software development. These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

In 1994, four authors Erich Gamma, Richard Helm; Ralph Johnson und John Vlissides published a book titled Design Patterns - Elements of Reusable Object-Oriented Software which initiated the concept of Design Pattern in Software development.

These authors are collectively known as Gang of Four (GOF). According to these authors design patterns are primarily based on the following principles of object orientated design.

* Program to an interface not an implementation
* Favor object composition over inheritance

Design Patterns have two main usages in software development.

* Common platform for developers

Design patterns provide a standard terminology and are specific to particular scenario. For example, a singleton design pattern signifies use of single object so all developers familiar with single design pattern will make use of single object and they can tell each other that program is following a singleton pattern.

* Best Practices

Design patterns have been evolved over a long period of time and they provide best solutions to certain problems faced during software development. Learning these patterns helps un-experienced developers to learn software design in an easy and faster way.

As per the design pattern reference book **Design Patterns - Elements of Reusable Object-Oriented Software**, there are 23 design patterns. These patterns can be classified in three categories: Creational, Structural and behavioral patterns. As shown in the table below, the pattern name and description of its work.

|  |  |
| --- | --- |
| **Pattern & Description** | **Description** |
| **Creational Patterns** | These design patterns provides 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. |
| **Structural Patterns**. | These design patterns concern class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities |
| **Behavioral Patterns**. | These design patterns are specifically concerned with communication between objects |
| **J2EE Patterns** | These design patterns are specifically concerned with the presentation tier. These patterns are identified by Sun Java Center. |

The patterns described in the table below are some of the most common, basic and important design patterns one can find in the areas of object-oriented design and programming. Some of these fundamental design patterns, such as the Interface, Abstract Parent, Private Methods, etc.,

|  |  |
| --- | --- |
| Pattern Name | Description of work done by the pattern |
| Interface | Can be used to design a set of service provider classes that offer the same service so that a client object can use different classes of service provider objects in a seamless manner without having to alter the client implementation. |
| Abstract Parent Class | Useful for designing a framework for the consistent implementation of the functionality common to a set of related classes. |
| Private Methods | Provide a way of designing a class behavior so that external objects are not permitted to access the behavior that is meant only for the internal use. |
| Accessor Methods | Provide a way of accessing an object’s state using specific methods. This approach discourages different client objects from directly accessing the attributes of an object, resulting in a more maintainable class structure |
| Constant Data Manager | Useful for designing an easy to maintain, centralized repository for the constant data in an application. |
| Immutable Object | Used to ensure that the state of an object cannot be changed. May be used to ensure that the concurrent access to a data object by several client objects does not result in race conditions. |

## Introduction to the java framework (MVC, struts, spring, JSF)

MVC - Model-View-Controller - is a design pattern for the architecture of web applications. It is a widely adopted pattern, across many languages and implementation frameworks, whose purpose is to achieve a clean separation between three components of most any web application, these three components are:

* Model: business logic & processing
* View: user interface (UI)
* Controller: navigation & input

In general, J2EE applications follow the MVC design pattern



### 2.3.1 Benefits of the MVC model

MVC benefits fall into three categories:

1. **Separation of concerns in the codebase**

Separation of Model, View, and Controller:

• allows re-use of business-logic across applications

• allows development of multiple UIs without touching business logic codebase

• discourages "cut-&-paste" repetition of code, streamlining upgrade & maintenance tasks

1. **Developer specialization and focus**

UI (User Interface) developers can focus exclusively on UI, without getting bogged down in business logic rules or code.

Business logic developers can focus on business logic implementation and changes, without having to slog through sea of UI widgets

1. **Parallel development by separate teams**

Business logic developers can build "stub" classes that allow UI developers to forge ahead before business logic is fully implemented. UI can be reworked as much as the customer requires without slowing down the development of code that implements business rules.

Business rule changes are less likely to require revision of the UI (because rule changes don't always affect structure of data the user sees).

## 2.4 How to implement selected design pattern in java

**Model Implementation** Technologies

Within the J2EE framework, there are a range of technologies that may be applied to implementation of the model layer. These include:

• Object-Relational Mapping Frameworks (Java Objects <-> Relational Databases):

• Hibernate, an open-source persistence framework

• Oracle's TopLink

• Sun's EJB (Enterprise Java Beans)

• Sun's JDO (Java Data Objects)

• Hand-coded data accessors via the JDBC API

The attributes and merits of each of these solutions are outside the scope of this presentation. In general, however, Object-Relational mapping frameworks involve a layer of abstraction and automation between the application programmer and data persistence, which tends to make application development faster and more flexible.

Differences between specific O-R Mapping technologies turn on the array of services offered by each technology, vendor specificity, and how well these differences map to a particular application's development and deployment requirements.

**View (Detail)**

Elements of the View:

The application's View is what the user sees. In a web application, the View layer is made up of web pages; in a web application that can be accessed by non-traditional browsers, such as PDAs or cell-phones, the View layer may include user interfaces for each supported device or browser type.

In any case, the View includes:

• Core data - the subject of a page's business

• Business logic widgets (e.g., buttons to perform Edit or Save)

• Navigation widgets (e.g., navigation bar, logout button)

• Skin (standard look of the site: logos, colors, footer, copyright, etc.)

**• View Technologies**

In the context of a web application, view technologies are used to render the user interface (UI) in a way that properly and dynamically links it to the application's business-logic and data layer (i.e., to the model).

**Java Server Pages** (JSPs) are a widely-utilized technology for constructing dynamic web pages. JSPs contain a mix of static markup and JSP-specific coding that references or executes Java. In brief:

• JSP Tags call compiled Java classes in the course of generating dynamic pages.

• JSP Directives are instructions processed when the JSP is compiled. Directives set page-level instructions, insert data from external files, and specify custom tag libraries.

• Java code - in sections called "scriptlets" - can be included in JSP pages directly, but this practice is strongly discouraged in favor of using JSP Tags

**XML Pipelining** is another technique for rendering the User Interface. Apache's Cocoon and Orbeon's OXF are technologies that use this technique. (XML pipelining simply means executing a series of transformations of XML documents in a proscribed order. For example, a data structure pulled from a persistent store may be rendered as an XML document, then - in a pipeline may be augmented with data from a separate data structure; sorted; and rendered in HTML format using a series of XSLT transformations. A simple example of an XML pipeline using Apache's Ant tool can be viewed in this article.)

**• Templates**

In order to maintain a consistent look-and-feel on a site, pages are often in template form.

The web page's <body> is where core business content of a page can be found. Using a template to render a consistent header, footer, and navigation UI around the core business content standardizes a site's graphic presence, which tends very strongly toward making users' experience smoother and less prone to error

**• Styling**

Cascading Style Sheets are a critical component of rendering a consistent look and feel in a web site (cf. this CSS Tutorial for more on Cascading Style Sheets). CSS is used to specify colors, backgrounds, fonts and font-size, block-element alignment, borders, margins, list-item markers, etc. By specifying the designer's choices in a single file (or group of files), and simply referring to the style sheet(s) in each web page, changes in style can be accomplished without altering the pages themselves.

**• The Decorator Pattern**

Templating and styling are aspects of an important "Gang of Four" design pattern, called the Decorator Pattern. Using the Decorator Pattern simply means wrapping some core object in something that gives it additional functionality. In the context of building a user interface for the web - the View layer of an MVC-architected application - this might mean wrapping:

• core business information in a

• template (e.g., header + footer + navigation), which includes (in the common

HTML header) reference to a

• CSS style sheet (where background, colors, fonts, etc. are specified)

**Controller Detail**

• Responsibilities of the Controller:

The Controller will do the following:

• Parse a user request (i.e., "read" it)

• Validate the user request (i.e., assure it conforms to application's requirements)

• Determine what the user is trying to do (based on URL, request parameters, and/or form elements)

• Obtain data from the Model (if necessary) to include in response to user

• Select the next View the client should see The sequencing of calls to the Model (business-logic layer), and/or the sequencing of views and required input from the user defines the application's workflow. Workflow is thus defined in the Controller layer of the application.

**• Controller Technologies**

There are numerous application frameworks that can be used to provide the control layer in an MVC-architected application.

These include:

• Struts

• Java Server Faces

• WebWork

• Spring

## 2.5 Chapter question

a) Explain why design patterns are important while designing software’s.

b) Explain the java design pattern; in addition explain the various types of java design patterns that can be used.

c) Explain how the MVC model is used in the java framework