## ASSIGNMENT

## ON

## Case Tool, Solid Principle

COURSE NAME: SYSTEM ANALYSIS AND DESIGN

COURSE CODE: CSE 326

SUBMITTED TO:

SAKIB KHAN

LECTURER

DEPARTMENT OF CSE

SUBMITTED BY:

ZULKAR NINE

163432601

43RD BATCH

DEPARTMENT OF CSE

SUBMITTED DATE: 07-11-2019

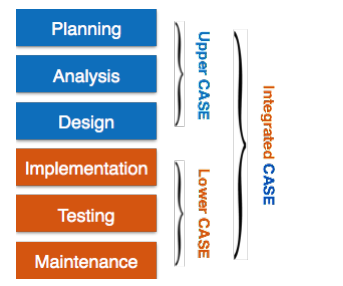
CASE Tools:

CASE tools are set of software application programs, which are used to automate SDLC activities. CASE tools are used by software project managers, analysts and engineers to develop software system.

There are number of CASE tools available to simplify various stages of Software Development Life Cycle such as Analysis tools, Design tools, Project management tools, Database Management tools, Documentation tools are to name a few.

Use of CASE tools accelerates the development of project to produce desired result and helps to uncover flaws before moving ahead with next stage in software development.

Components of CASE Tools:



Upper Case Tools - Upper CASE tools are used in planning, analysis and design stages of SDLC.

Lower Case Tools - Lower CASE tools are used in implementation, testing and maintenance.

Integrated Case Tools - Integrated CASE tools are helpful in all the stages of SDLC, from Requirement gathering to Testing and documentation.

Case Tools Types:

Diagram tools:

These tools are used to represent system components, data and control flow among various software components and system structure in a graphical form.

For example, Flow Chart Maker tool for creating state-of-the-art flowcharts.

Process Modeling Tools:

Process modeling is method to create software process model, which is used to develop the software. Process modeling tools help the managers to choose a process model or modify it as per the requirement of software product.

For example, EPF Composer

Project Management Tools:

These tools are used for project planning, cost and effort estimation, project scheduling and resource planning. Managers have to strictly comply project execution with every mentioned step in software project management. Project management tools help in storing and sharing project information in real-time throughout the organization.

For example, Creative Pro Office, Trac Project, Basecamp.

Documentation Tools

Documentation in a software project starts prior to the software process, goes throughout all phases of SDLC and after the completion of the project. Documentation tools generate documents for technical users and end users. Technical users are mostly in-house professionals of the development team who refer to system manual, reference manual, training manual, installation manuals etc. The end user documents describe the functioning and how-to of the system such as user manual.

For example, Doxygen, DrExplain, Adobe RoboHelp for documentation.

Analysis Tools

These tools help to gather requirements, automatically check for any inconsistency, inaccuracy in the diagrams, data redundancies or erroneous omissions. For example, Accept 360, Accompa, CaseComplete for requirement analysis, Visible Analyst for total analysis.

Design Tools

These tools help software designers to design the block structure of the software, which may further be broken down in smaller modules using refinement techniques. These tools provides detailing of each module and interconnections among modules. For example, Animated Software Design Configuration Management Tools An instance of software is released under one version.

Configuration Management tools

An instance of software is released under one version. Configuration Management tools deal with –

* Version and revision management
* Baseline configuration management
* Change control management

Programming Tools

These tools consist of programming environments like IDE Integrated Development Environment, in-built modules library and simulation tools. These tools provide comprehensive aid in building software product and include features for simulation and testing.

For example, Cscope to search code in C, Eclipse.

Prototyping Tools

Software prototype is simulated version of the intended software product. Prototype provides initial look and feel of the product and simulates few aspect of actual product. Prototyping CASE tools essentially come with graphical libraries. They can create hardware independent user interfaces and design. These tools help us to build rapid prototypes based on existing information. In addition, they provide simulation of software prototype.

For example, Serena prototype composer, Mockup Builder.

Web Development Tools

These tools assist in designing web pages with all allied elements like forms, text, script, graphic and so on. Web tools also provide live preview of what is being developed and how will it look after completion.

For example, Fontello, Adobe Edge Inspect, Foundation 3, Brackets.

Maintenance Tools

Software maintenance includes modifications in the software product after it is delivered. Automatic logging and error reporting techniques, automatic error ticket generation and root cause Analysis are few CASE tools, which help software organization in maintenance phase of SDLC.

For example, Bugzilla for defect tracking, HP Quality Center.

SOLID

What is SOLID?

S -SRP: Single Responsibility Principle

O– OCP: Open-Closed Principle

L – LSP: Liskov Substitution Principle

I– ISP: Interface Segregation Principle

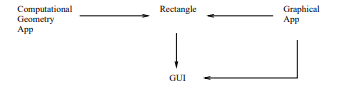
D– DIP: Dependency Inversion Principle

• Basically a set of principles for object-oriented design (with focus on designing the classes.

Single Responsibility Principle:

A class should have one and only one reason to change.

SRP: Single Responsibility Principle



• Example: Rectangle class with draw() and area()

• Computational geometry now depends on GUI, via Rectangle.

• Any changes to Rectangle due to Graphical application necessitates rebuild, retest, etc. of Comp. geometry app.

SRP: Summary

• “SRP is the simplest of the principles, and one of the hardest to get right.”

• We tend to join responsibilities together.

• SRP says we need to go against this tendency

Open/Closed Principle

Software entities should be open for extension, but closed for modification.

OCP: Open-Closed Principle

• “All systems change during their life cycles.” (Ivar Jacobson).

• “Software entities should be open for extension, but closed for modification.” (variation on Bertrand Meyer’s idea).

• Goal: avoid a “cascade of changes to dependent modules”.

• When requirements change, you extend the behavior, not changing old code.

OCP: Summary

• OCP is “at the heart of OOD”.

• Simply using an OOP is not enough: Need dedication to apply abstraction.

• OCP can greatly enhance reusability and maintainability.

Liskov Substitution Principle

A subclass should behave in such a way that it will not cause problems when used instead of the superclass.

LSP: Liskov Substitution Principle

• “Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.” (original idea due to Barbara Liskov).

• Violation means the user class’s need to know ALL implementation details of the derived classes of the base class.

• Violation of LSP leads to the violation of OCP.

LSP: Summary

• LSP is an important property that holds for all programs that conform to the Open-Closed principle.

• LSP encourages reuse of base types, and allows modifications in the derived class without damaging other components.

Interface Segregation Principle

Clients should not be forced to depend upon interfaces that they don't use.

ISP: Interface Segregation Principle

• “Clients should not be forced to depend upon interfaces that they do not use.”

• Avoid “fat interfaces”.

• Fat interfaces: interfaces of a class that can be broken down into groups that server differnt set of clients. • Clients depending on a subset of interfaces need to change when other clients using a different subset changes.

ISP: Summary

• Should avoid interfaces that are not specific to a single client.

• Fat interfaces cause inadvertant coupling between unrelated clients.

Dependency Inversion Principle

High-level modules should not depend on low-level modules. Both should depend on abstractions.

DIP: Dependency Inversion Principle

• “A. High level modules should not depend upon low level modules. Both should depend upon abstractions.”

• “ B. Abstractions should not depend upon details. Details should depend upon abstractions.”

• DIP is an out-growth of OCP and LSP.

• “Inversion”, because standard structured programming approaches make the higher level depend on lower level.

DIP: Summary

• DIP promises many benefits of OO paradigm.

• Reusability is greately enhanced by DIP.

• Code can be made resilient to change by using DIP.

• As a result, code is easier to maintain.