Unit 20: Applied Programming and Design Principles

Unit code	**T/618/4902**
Unit level	**5**
Credit value	**15**

Introduction

The advanced features of programming languages are used to develop software that is efficient, robust and can be mathematically proven to work. Well-designed code can positively impact the performance of an application as well as the readability and extensibility of the code, thereby improving productivity and reducing cost.

Effective object orientated programming (OOP) should have low coupling, high cohesion and strong encapsulation, which is something that the SOLID principles help to obtain. The idea is that by applying those principles together, it makes it easier to write better quality code with greater diversity and robustness. The system created becomes easy to maintain, to reuse and to extend over time. SOLID principles help software developers to achieve scalability and avoid creating code that breaks every time it needs a change. Clean coding maintains the readability of the programs produced by encouraging descriptive naming of objects and keeping to a single purpose model for each entity. Programming patterns work to ensure that designs produced are language independent, encapsulate ideas and are reusable in multiple circumstances.

The development of an application to process a large data set is a practical example of how to solve a problem that can be used in many different situations, can help deepen the understanding of OOP and help improve software design and reusability.

The aim of this unit is to familiarise students with these concepts and their best practices to ensure that their code is in line with industry standards. Among the topics included in this unit are object-orientated programming, introduction to design patterns and SOLID, including its version of five principles of object-oriented programming and automated software testing.

The unit is especially useful for those intending to move into computer science, software development, programming, systems analysis and software testing

Learning Outcomes

By the end of this unit, students will be able to:

LO1 Investigate the impact of SOLID development principles on the OOP paradigm

LO2 Design a large dataset processing application using SOLID principles and clean coding techniques

LO3 Build a data processing application based on a developed design

LO4 Perform automatic testing on a data processing application.

Essential Content

LO1 Investigate the impact of SOLID development principles on the OOP paradigm

Object-orientated programming (OOP) paradigm characteristics:

Understand the OO characteristics and their application in developing code, including encapsulation, polymorphism, constructors and destructors, sub-objects, abstraction, interface, method overriding and redefinition, templates and containers.

Object-orientated class relationships:

Understand the elements of the OO relationships, including generalisation and inheritance, realisation, dependency, aggregation and composition.

Design patterns:

Aims and benefits of reusable design patterns, e.g. general reusable solution, represent an idea and language independent.

Grouping of design patterns into creational, structural and behavioural groups.

Clean coding techniques:

Simple design, e.g. keeping configurable data at high levels, polymorphism, consistency in methods, meaningful variable and constant names and encapsulate boundary conditions.

Creating small functions by including single action, minimal parameters, descriptive names, comments to explain code and warn of consequences.

Structure source code to separate concepts vertically, declare variables close to usage and keep lines short.

Develop objects and data structures for one action so that they are small.

Understand why non-static methods are preferable to static methods.

Design tests to ensure they are readable, fast and independent.

Understand bad test design, e.g. rigid, fragile, immobile, complex, repetitive.

SOLID design principles:

Understand and apply the component parts of SOLID design principles to make software understandable, flexible and maintainable, including Single-responsibility principle, Open-closed principle, Liskov substitution principle, Interface segregation principle and Dependency inversion principle.

LO2 **Design a large dataset processing application using SOLID principles and clean coding techniques**

Large datasets (public domain):

Design of application that can accommodate pre-existing large (500+) datasets, e.g. list of members of parliament, register of members' interests (Commons and Lords), list of public domain films (e.g. in the USA), list of public domain books, list of public domain music.

Data structures:

Use of data structures in application development, e.g. stack, array, multi-dimensional array, set, queue, list and linked list.

Apply tree types, including active, passive and recursive.

Operations:

Use of operations in application development, e.g. hash functions and pointers.

Utilise sorts, e.g. insertion, quick, merge and heap.

Utilise searches, e.g. linear, binary tree and recursive.

LO3 **Build a data processing application based on a developed design**

Implementation:

Utilise an appropriate language and development tools, incorporate security and maintainability expectations.

Produce program code that implements a design based on SOLID principles, clean coding techniques and programming patterns.

Understand and interpret design features, meet requirements, input, output, processing security, portability and maintainability.

LO4 **Perform automatic testing on a data processing application**

Types of automatic testing:

Understand the uses of automation in setting up regression tests, data set up generation, product installation, GUI interaction, defect logging, unit testing and integration testing of main application.

Tool automation parameters:

Understand the meaning of data driven capabilities, debugging and logging capabilities, platform independence, extensibility and customisability, email notifications, version control friendly.

How automated testing features support unattended test runs.

Understanding of testing logic and updates code to make testing easier through the use of stubbing/patching

Follow common testing frameworks and methodologies:

Understand the circumstances where different frameworks perform best, Data Driven Automation Framework, Keyword Driven Automation Framework, Modular Automation Framework and Hybrid Automation Framework.

Tools:

Investigate a range of tools that are commercially available.

Functional, e.g. QuickTest Professional (HP), Coded UI (Microsoft), Selenium, Open IT (open source).

Non-functional, e.g. LoadRunner (HP), JMeter (Apache), Burp Suite (PortSwigger).

Self-built testing tools:

Investigate the value of developer-designed and built tools to test features and functions of a specific application.

Learning Outcomes and Assessment Criteria

Pass	Merit	Distinction
LO1 Investigate the impact of SOLID development principles on the OOP paradigm		
P1 Investigate the characteristics of the object-orientated paradigm, including class relationships and SOLID principles. **P2** Explain how clean coding techniques can impact on the use of data structures and operations when writing algorithms.	**M1** Analyse, with examples, each of the creational, structural and behavioural design pattern types.	**D1** Evaluate the impact of SOLID development principles on object-orientated application development.
LO2 Design a large dataset processing application using SOLID principles and clean coding techniques		
P3 Design a large data set processing application, utilising SOLID principles, clean coding techniques and a design pattern. **P4** Design a suitable testing regime for the application, including provision for automated testing.	**M2** Refine the design to include multiple design patterns.	

Pass	Merit	Distinction
LO3 Build a data processing application based on a developed design		**D2** Analyse the benefits and drawbacks of different forms of automatic testing of applications and software systems, with examples from the developed application.
P5 Build a large dataset processing application based on the design produced.	**M3** Assess the effectiveness of using SOLID principles, clean coding techniques and programming patterns on the application developed.	
LO4 Perform automatic testing on a data processing application		
P6 Examine the different methods of implementing automatic testing as designed in the test plan. **P7** Implement automatic testing of the developed application.	**M4** Discuss the differences between developer-produced and vendor-provided automatic testing tools for applications and software systems.	

Recommended Resources

Textbooks

CLARKE JILL, (2020) *Software Developer*, London: BCS

FISHPOOL B & FISHPOOL M, (2020) *Software Development in Practice*, BCS

FREEMAN E., FREEMAN E., SIERRA K., BATES B. (2004) *Head First Design Patterns*, London: O'Reilly

GAMMA E, HELM R, JOHNSON R, VLISSIDES J, (1994) *Design Patterns: elements of reusable object-oriented software*, Addison Wesley

MARTIN, RC, (2017) *Clean Architecture: A Craftsman's Guide to Software Structure and Design.* London Pearson, Addison Wesley

Journal

academic.oup.com	*Oxford Academic*
	ITNow: British Computer Society
	(general reference)

Web

baeldung.com	Baeldung
	A Solid Guide to SOLID Principles
	(general reference)
tutorialspoint.com	*Software Testing Dictionary*
	(general reference)

Links

This unit links to the following related units:

Unit 1: Programming

Unit 7: Software Development Lifecycles

Unit 19: Data Structures and Algorithms

Unit 22: Application Development

Unit 24: Advanced Programming for Data Analysis