

|  |  |
| --- | --- |
| Course / Programme: | BEng (Hons) Software Engineering |
| Module name and code: | Introduction to Software Development (SWE4201) |
| Student ID | 2116149 |
| Tutor: | Abdul Razak |
| Assignment Number: | 1 |
| Assignment Title: | The Friston College |
| Weighting | 60% of overall module grade |
| Issue Date: | 20 March 2023 |
| Submission Deadline: | 12 May 2023@ 1600 |
| Learning Outcomes:   1. Develop algorithms to solve given problems. 2. Construct logically and syntactically correct programs using appropriate programming constructs. 3. Demonstrate systematic testing of programmed solutions to identify and correct syntax, semantic and logical errors. 4. Present effective use of modelling standards to represent system structure, behaviour and interaction. | |

Contents

[Task 1 - Object Oriented Design Theory 1](#_Toc157805434)

[Task 2 - Develop a programmed solution for the college 4](#_Toc157805435)

[Program Coding 4](#_Toc157805436)

[Student Class 4](#_Toc157805437)

[Course Class 6](#_Toc157805438)

[Enrolment Class 8](#_Toc157805439)

[College Class 10](#_Toc157805440)

[Test Class 23](#_Toc157805441)

[UML Diagram 26](#_Toc157805442)

[Task 3 - Discuss the challenges relating to applying software development techniques in a business context. 27](#_Toc157805443)

[Bibliography 29](#_Toc157805444)

# Task 1 - Object Oriented Design Theory

The situation includes a community college in the Friston area that is in danger of closing owing to budget cuts. The objective at hand is to produce the computer programme that the college requested be made in order to increase its productivity. The choice to base the program's solution on the Object-Oriented paradigm was decided during the initial design process. The first task's main objective is to provide an overview of the fundamental concepts of object-oriented programming, including encapsulation, abstraction, inheritance, polymorphism, class hierarchies, and interface programming. This information will be helpful in creating the computer programme for the college utilising an object-oriented methodology.

One of the core ideas of object-oriented programming is **encapsulation**. It is the process of restricting access to an item to its public interface and concealing all of its internal components. Encapsulation makes sure that an object's state cannot be accessed directly from outside the object, which can assist prevent unintentional state modification. An object's internal state is often concealed from external code and is only accessible through its public interface, which normally consists of a set of methods that may be used to interact with the object. Modular programming is made possible through encapsulation, which allows changes to an object's internal state to be done without having an impact on other programme elements that depend on the object's visible interface.

Another key idea in object-oriented programming is **abstraction**. It is the process of determining and modelling an object's core features while ignoring its optional elements. Programmers can develop models of complex systems that are simple to comprehend and modify by using abstraction. Because abstract models can serve as the foundation for numerous implementations, it also makes it easier to reuse code. Abstract classes or interfaces, which establish a standard contract for the implementation of objects with comparable behaviours, are frequently used to achieve abstraction.

A process called **inheritance** enables a class to take on traits and behaviours from another class. Inheritance is used to build a hierarchy of classes, with the base class often acting as the root class for all the classes that derive from it. By letting classes to take on traits and behaviours from parent classes, inheritance facilitates code reuse and lessens duplication. Additionally, it enables courses to customise and hone the qualities and behaviours of their parent classes to better suit their individual requirements. It is possible to organise code using inheritance for improved scalability and maintainability as well as to simulate complicated systems.

**Polymorphism** is a fundamental idea in object-oriented programming that allows objects of various classes to be treated as if they were of the same class. As it permits generic object manipulation without knowledge of an object's exact type, polymorphism allows for flexibility and extensibility in the design of programmes. By using inheritance or interfaces, which let several objects implement the same set of methods in various ways, polymorphism can be accomplished. It enables the interchangeability of objects from other classes inside a programme, which can streamline the logic of the programme and increase the modularity of the code.

**Classes** can be arranged in a hierarchical manner using class hierarchies. The base class, which forms the basis for all classes that derive from it, is often the most generic class. Classes can specialise in and improve the traits and behaviours of their parent classes through inheritance, while also gaining access to their shared traits and behaviours. Class hierarchies can be used to organise code for improved maintainability and scalability as well as to simulate complicated systems. Additionally, they are useful for putting into practise design patterns, which are reusable solutions to typical programming issues.

**Programming to interfaces** is a method of design that promotes the usage of interfaces to provide a shared contract between things. A set of required methods are specified by an interface, but the specifics of how those methods must be implemented are not. Thus, polymorphism and flexibility in the design of the programme are made possible by allowing many classes to offer their own implementations of the same interface. Modularity, reuse, and extensibility of code can all be increased by programming to interfaces. Additionally, because interfaces clearly specify an object's behaviour, they can make testing and debugging easier.

In summary, encapsulation, abstraction, inheritance, and polymorphism are the cornerstones of the object-oriented programming paradigm. Encapsulation includes obscuring an object's innermost workings and restricting access to its visible surface. Finding and modelling an object's essential qualities while ignoring its optional elements is the process of abstraction. One class can take on the characteristics and behaviours of another class through a method called inheritance. Objects of various classes can be treated as though they belong to the same class thanks to polymorphism. Class hierarchies set up classes according to an inheritance-based hierarchy. Using interfaces to provide a common contract between objects is encouraged by the design philosophy known as "programming to interfaces." By following these guidelines, programmers can build well-structured, scalable, and maintainable applications that can change along with their surroundings and requirements.

# Task 2 - Develop a programmed solution for the college

## Program Coding

### Student Class

A picture containing text, screenshot, software

Description automatically generated

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

### Course Class

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

### Enrolment Class

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

### College Class

A screenshot of a computer program

Description automatically generated with medium confidence

A picture containing text, screenshot

Description automatically generated

A picture containing text, screenshot

Description automatically generated

A picture containing text, screenshot, software, multimedia software

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidenceA picture containing text, software, multimedia software, screenshot

Description automatically generatedA screen shot of a computer

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidenceA screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidenceA screen shot of a computer program

Description automatically generated with low confidence

A picture containing text, screenshot, software

Description automatically generatedA screen shot of a computer

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with low confidence

A screenshot of a computer program

Description automatically generated with medium confidence

A screen shot of a computer program

Description automatically generated with low confidence

A picture containing text, screenshot

Description automatically generatedA screen shot of a computer program

Description automatically generated with low confidence

A screenshot of a computer program

Description automatically generated with medium confidenceA screen shot of a computer program

Description automatically generated with medium confidence

A picture containing text, screenshot, software

Description automatically generatedA screen shot of a computer program

Description automatically generated with low confidence

### Test Class

A screenshot of a computer program

Description automatically generated with medium confidence

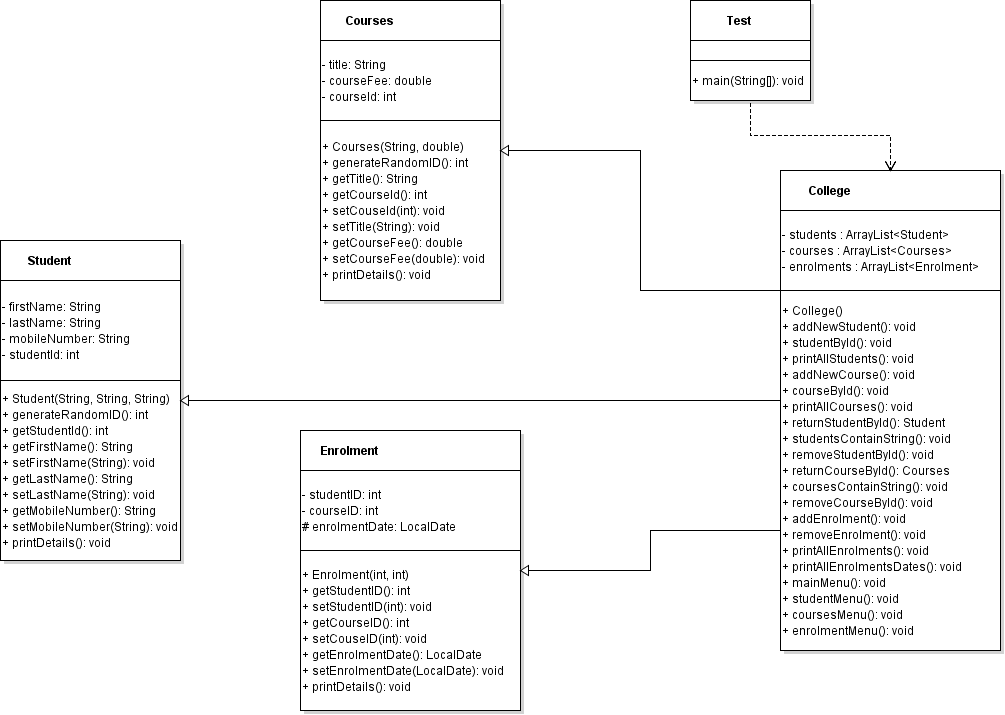
A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

## UML Diagram



# Task 3 - Discuss the challenges relating to applying software development techniques in a business context.

As firms work to streamline their procedures, increase consumer engagement, and outperform rivals, software development is a vital component of contemporary corporate operations. But because of many issues, such as organisational culture, resource limitations, and technological limitations, applying software development approaches in a business setting can be difficult. of this essay, using pertinent academic sources, I'll talk about the difficulties of using software development approaches in a corporate setting.

The requirement to integrate technology with organisational culture and values is one of the main obstacles to using software development methodologies in a commercial setting. Organisations have distinctive cultures that influence how they see innovation, risk management, and the adoption of new technologies. Successful technology adoption, according to Brynjolfsson and McAfee (2014), necessitates a cultural transformation that integrates technology with the organization's strategic goals, values, and procedures. This can be difficult in practise, though, as organisations may have to deal with internal opposition, power struggles, and communication difficulties. For instance, a business can encounter opposition from staff members who worry that technology would eliminate their jobs or change their working circumstances.

Applying software development approaches in a corporate setting presents another challenge: balancing resource limitations with project scope and quality. Resource-intensive software development projects may necessitate large expenditures on technology, software, and human resources. Managing software development projects necessitates striking a precise balance between project scope, quality, and resources, claim Boehm and Turner (2005). Resource limitations in the context of a business can affect the scope and quality of software development initiatives, which might result in subpar results. To fulfil a strict schedule or budget, a corporation could need to decrease the testing and quality assurance procedure or limit the functionalities of a software product.

The necessity to overcome technology constraints and developments is a third difficulty of using software development approaches in a commercial environment. New technologies and techniques are constantly being developed in the world of software development. Succi et al. (2019) assert that the success of software development projects depends on remaining current with technological developments and selecting the appropriate technology stack. For firms without the infrastructure or technical know-how to take advantage of the newest technology, this can be difficult. Technology developments may also bring up brand-new dangers and difficulties, like cybersecurity issues, legal needs, and moral dilemmas. A business may need to weigh the advantages of cloud computing against the dangers of data breaches and privacy violations, for instance.

In conclusion, there are a number of reasons why using software development methodologies in a commercial setting might be difficult, including organisational culture, resource limits, and technology constraints. A cultural change is necessary for successful technology adoption in order to match technology with the organization's strategic goals, beliefs, and operating procedures. Project scope, quality, and resource management are all delicately balanced in software development projects. The success of software development projects depends on keeping up with technical developments and selecting the appropriate technology stack. All stakeholders, including corporate executives, programmers, and end users, must be involved in a collaborative and agile strategy to address these difficulties.

# Bibliography

Barnes, D. J., & Kölling, M. (2012). Objects first with Java: A practical introduction using BlueJ. Pearson Education.

Booch, G. (1994). Object-oriented design with applications. Benjamin-Cummings Publishing Company.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design patterns: elements of reusable object-oriented software. Addison-Wesley.

Oracle. (n.d.). The Java Tutorials: Object-Oriented Programming Concepts. Retrieved May 12, 2023, from https://docs.oracle.com/javase/tutorial/java/concepts/

Shalloway, A., & Trott, J. R. (2019). Design patterns explained: a new perspective on object-oriented design. Addison-Wesley.

Wu, J., & Lu, L. (2019). Object-oriented programming and design: a practical approach. Springer.

Brynjolfsson, E., & McAfee, A. (2014). The second machine age: Work, progress, and prosperity in a time of brilliant technologies. WW Norton & Company.

Boehm, B., & Turner, R. (2005). Management challenges to implementing agile processes in traditional development organizations. IEEE Software, 22(5), 30-39.

Succi, G., Marchesi, M., & Kim, S. (2019). Agile software development. In Agile software development (pp. 1-20). Springer.

O'Connor, R. V., & Laporte, C. Y. (2012). Guest editorial: Software process improvement in the small. Journal of Systems and Software, 85(4), 793-794.

Brown, K. A., & Loch, C. H. (2018). Enabling innovation through rigorous project portfolio management. California Management Review, 60(1), 26-49.

Mäenpää, K., Järvi, A., & Toivonen, J. (2018). Developing software in distributed teams: A systematic review. Journal of Systems and Software, 146, 177-201.

Lycett, M., Macredie, R., Patel, C., & Paul, R. J. (2015). Developing sustainable business models for pervasive information systems. MIS Quarterly, 39(1), 217-236.