**Introduction to Software Development**

**(SWE4201)**

**Portfolio Report**

**Module Code: SWE4201**

**Year/Semester: 2023-24 / Semester 2**

**Abstract**  
In this report the design, implementation and testing of three programing tasks belonging to the SWE4201 module were completed. The tasks serve as examples of using Python and Java as software engineering tools to solve specific problems in the real-world. Such programming assignments are the highlights of this report due to their systematic way of handling programming, the obstacles faced during coding, and the knowledge gained afterwards. These concepts have been bolstered with academic literature further.

**Introduction**

The advancement of software systems is instrumental in the identification and resolution of problems in a wide variety of fields. The portfolio in this report contains answers to three tasks, that require the students to focus on algorithm design, appropriate programming constructs, use of structured testing methodologies, and the OOP principles. The report also considers the obtained learning outcomes and reflects on the learning process with regard to the successes and the failures.

**Task 1: Student Registration System (Python Program)**

**Objective**

Develop a Python program to manage student registrations, using modular programming, error handling, and a robust data structure.

**Design and Methodology**

The student registration system was designed with the following principles:

* **Modularity:** Functions for core operations like adding, retrieving, and deleting records were implemented separately for maintainability.
* **Data Structure:** A dictionary was chosen for its efficient key-value pair management, allowing quick lookups by student ID.
* **Error Handling:** try-except blocks were integrated to handle exceptions such as duplicate entries and invalid input.
* **User Interaction:** Input validation ensured that user-provided data conformed to expected formats.

**Key Design Decisions**

1. **Data Persistence:** While the current implementation uses in-memory storage, future iterations could extend this to persistent databases.
2. **Error Messages:** User-friendly messages were added to ensure better interaction.

**Code Explanation**

The program contains the following key components:

* **Functionality for Adding Records:**
  + Prevents duplicate entries by validating the student ID.
* **Functionality for Data Retrieval:**
  + Retrieves either a specific student record or all records using conditional checks.

**Expanded Code Snippet:**

**Challenges**

* Balancing modularity with functionality: Ensuring the program was not overcomplicated while remaining modular.
* Validating user input to avoid runtime errors.

**Testing and Results**

* **Test Scenarios:** Edge cases such as attempting to add duplicate records and retrieving non-existent student IDs were tested.
* **Outcome:** The program performed as expected in all scenarios.

**Output Placeholder**

* *Insert screenshots of the program output.*

**Task 2: Structure Display Program (Python Program)**

**Objective**

Design a Python program to display geometric structures, demonstrating the use of loops and conditional logic.

**Design Approach**

The program was structured to handle multiple patterns, ensuring scalability for future enhancements:

* **Reusable Functions:** Separate functions for each structure were implemented.
* **Customizable Input:** Users can specify the size of the structure, with validation to prevent invalid entries.
* **Efficiency:** Nested loops ensured minimal computational overhead.

**Key Design Decisions**

1. **Scalability:** Designed the program to allow additional structures with minimal changes.
2. **Error Handling:** Managed edge cases, such as zero or negative inputs.

**Expanded Code Snippet:**

**Challenges**

* Ensuring uniform alignment for all patterns.
* Handling large inputs without impacting performance.

**Testing and Results**

* **Test Scenarios:** Patterns were tested with small and large sizes to ensure alignment and accuracy.
* **Outcome:** The program successfully generated all patterns with correct formatting.

**Output Placeholder**

* *Insert diagrams of displayed patterns (e.g., triangles, pyramids).*

**Task 3: Student Record System (Java Program)**

**Objective**

Create a Java-based student record system incorporating object-oriented principles.

**Requirements Analysis**

The system was designed to manage the following:

* **Student Details:** Names, IDs, courses, and departments.
* **Operations:** Adding, updating, and retrieving records.
* **Object-Oriented Principles:** Encapsulation, inheritance, and modular design were emphasized.

**Key Classes Identified:**

1. **Student:** Manages individual student data.
2. **Department:** Represents departmental operations.
3. **University:** Serves as a parent class.

**Design**

The design adheres to UML standards and includes a class diagram to illustrate relationships.

**Class Diagram Placeholder**

* *Insert a diagram showing relationships between Student, Department, and University.*

**Code Explanation**

* **Encapsulation:** Private attributes and public getter/setter methods restrict direct access to variables.
* **Inheritance:** Shared attributes like departmentName are inherited from the University class.

**Expanded Code Snippet:**

**Challenges**

* Designing a scalable and reusable class hierarchy.
* Managing input validation for complex data structures.

**Testing and Results**

* **Test Scenarios:** Focused on adding, updating, and retrieving records, as well as inheritance testing.
* **Outcome:** The system demonstrated consistent functionality.

**Output Placeholder**

* *Insert screenshots or logs showing test results.*

**Critical Reflection**

This assignment provided a comprehensive learning experience in software development. Below are the key reflections:

**Strengths**

1. **Algorithm Design:** Improved ability to break down problems into manageable components.
2. **Programming Constructs:** Gained proficiency in modular programming and object-oriented principles.
3. **Testing Practices:** Developed a systematic approach to testing, ensuring robust solutions.

**Areas for Improvement**

1. **Time Management:** Allocating more time for documentation and advanced testing.
2. **Database Integration:** Exploring persistent data storage options for future projects.

**Challenges**

* Understanding the nuances of error handling in Python and Java.
* Balancing functionality with simplicity in system design.

**Key Takeaways**

* The importance of modularity for maintainable and scalable systems.
* The value of systematic testing in ensuring reliable software.

**References**

1. Larman, C. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Pearson Education.
2. Eckel, B. (2006). *Thinking in Java*. Prentice Hall.
3. McConnell, S. (2004). *Code Complete: A Practical Handbook of Software Construction*. Microsoft Press.
4. Hunt, A., & Thomas, D. (1999). *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley.
5. Sommerville, I. (2015). *Software Engineering* (10th ed.). Pearson Education.
6. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
7. Dierbach, C. (2012). *Introduction to Computer Science Using Python: A Computational Problem-Solving Focus*. Wiley.