# Reflection on Object-Oriented Information Systems Module

**Learning Outcomes**

- Appraise and evaluate critically the concepts and principles of information systems.
- Design or modify and document an object-oriented information system using appropriate tools.
- Develop an object-oriented information system design, implementing this knowledge in applicable programming languages, such as Python and SQL.
- Develop, implement and evaluate critically information system solutions to facilitate business decisions.

**Evaluation of unit 7 and unit 11**

There were two assignments in Unit 7 and Unit 11, designing a self-checkout system for a supermarket and implementing a clinic appointment system. For the first assignment, I used a traditional OOP analysis to get the design. Based on the relationship between objects and workflow to get appropriate members and methods, there could be some managers to store the same things. I didn't get many obscure during the design, and I think I have experienced a similar system.

However, I spent a lot of time considering what pattern was the best choice for the second assignment. I also find some difficult to imagine a workflow. Finally, I picked an event-driven style based on analysing what activities could happen in the system, but I felt I could do better in time management; I took too much time to seek the "best" appropriate way to implement.

The main things I learned through these practices are more understanding of clients' requirements. Change in other words, your analysis of how the user could use your system is more detailed, and your design is more accurate or easy to implement. You understand more context or domain knowledge, and the analysis is more effective. And don't try to find a perfect solution at starting. Implementation in the iteration is a better style.

**Learned from Unit 1 to Unit 6:**

These units focus on what is an information system(IS), using an object-oriented(OO) way to analyse it and design a solution, and UML can help us during the design stage. Even if a programmer was involved in many systems development, they could never consider what an information system is. Maybe they want to finish implementing classes/functions/modules from requirement documents. However, this thinking way is too narrow. Evaluating the quality of a system is not just connecting software or hardware. It could refer to many non-technical parts, e.g. document integrity and user training, which can also decide whether the software is working or failing. The target of all the activities around a system is to let a system work as expected, and people can benefit from it. In addition, when I went through further readings, I felt that only relay technical tries to resolve all potential problems system faced is mostly impossible. When you have a complete picture, you can use a different view to review an issue and prompt a more effective way to fix it, not just get stuck in some specific areas.

During participating discussion on the failed system analysis and what disaster could be taken, many examples showed a failed system could be caused by various reasons, from misunderstanding user requirements to wrong operations of users. You can read more detail here.

OO thinking way on assisting client's requirement is helpful. From lectures, seminars, and further readings, I learned OO is a very subject way to design a system, and there could be a thousand ways to interpret a requirement. A system design depends on what consideration you take, and you need a science guide or effective method to verify your work. OOP principles aim to easy maintenance, reduce complexity, cutting reliability between components, and it also could bring some tradeoffs. There has no solution that can satisfy all cases. More importantly, taking a regular time to review design, update documents, summarise changing patterns, and refactor/iterate outdated code can significantly improve system quality.

UML is a useful tool for organising your thoughts and communicating with other stakeholders during a system design. I could say I didn't pay much attention to UML in my daily job. However, after studying in these units, I realised UML is good at examining your thinking; a clear UML could show a design without misunderstanding, and in which you can more easily calculate complexity between modules/objects. A clear UML graphic could be worth 30 minutes of explaining with talking, and it is revealed how your perspective on problems. I have tried to draw a ULM first before I need to discuss/clarify with others how the software should work. The discussion could be more productive with UML. For example,  we can ignore unnecessary details and concentrate on more essential parts, like whether the object relation or action order was appropriate.

An IS that was from a rough idea to get finally deployed could face countless decisions. The skill/knowledge/principles learned from these units will guide me to find the pathway that leads to creating a robust, meeting the real needs system.

**Learned from Unit 7 to Unit 10:**

Units highlight essential concepts and ideas about the database(DB), especially knowledge of relational form. Before the module, I often used DB when coding; however, I rarely considered its theory and had some incorrect thoughts about DB. I learned why normalisation for a relational DB is vital, and understanding it fixed me a wrong way of using DB. For example, I had updated data connected to multiple info in one column, and I just considered all the data I could parse in the code; the DB was just a simple container. This false thinking could lead to messy data information in the IS.

Combining relational DB and OOP thinking to design a system, I discovered that DB design could reflect people's consideration of the relations between system components. The OO analysis also has a considerable effect on the DB table structure. I think an ideal DB design and OOP share some philosophy: prefer single responsibility, from details data to extract abstraction (table generalization) or customize objects from a familiar concept (table specialization).

Big data rasing indicates that as various data sources introduce the IS, people can't predicate data type, size or volume the system could process. Dynamic and flexible are weaknesses of relational DB. Through participating in discussions about NoSQL, I learned why NoSQL is to be popular. Instead of replacing relational DB, NoSQL is meant to improve the weakness of the traditional relational DB. The

discussion is here.I felt more different knowledge you get, and you have more views/choices to help consider your system design.

**Summary and Further Consider**

During the course, as I studied more, I recognised I had some misunderstandings about system design/development. Even though I have some hands-on experience with OOP, I was still struggling with something, and it revealed that "how" could not answer "why"; I had not known the true facts behind concepts. The different activities and assignments helped improve my critical thinking and technical judgment skills as well as take various perspectives on the IS. I could not say my work is perfect, but the knowledge I gained could as my solid fundamental for future deeper learning.

As computer science advances, new systems could be evolved more complex, like big data and AI starting to become a necessary component of the information system to support human activity. Meanwhile, as software as a service (SaaS) becomes the mainframe, it brings many new concepts, like microservice, service-oriented architecture(SOA), etc. I think the thinking and analysis way are essential, and it does not matter what fancy concept or name I know, but vital is I understand the reason and facts behind them. Each technology has its position. As I progress through the course, I hope not to be satisfied with these skills but improve them also. I look forward today I learned can inspire me in future learning.