Work Load Participation Index Form

for

Assignments & Project(s)

(To be Submitted/Attached with Every Assignment and Phases of Project)

(Date of Submission): April 20nd 2020

Artifact/Document Type (e.g. Exam/Assignment #1/#2/#3, Project Phase 1/2): Final Exam
Participation in his/her allocated task's completion

5= Full (as allocated) 4= Partial (slightly less) 3=Half (as allocated) 2/1=Little 0= No Participation

Student Number	*Student Name*	*Participation Index Value e.g. 5/4/3/2/1*	*Signature*
040946430	*Thang Nguyen*	*5*	*Thang Nguyen*
040953846	*Diep Pham*	*5*	*Diep Pham*
040950904	*Mukta Debnath*	*5*	*Mukta Debnath*
040951042	*Ningxin Zhao*	*5*	*Ningxin Zhao*
040930563	*Min Li*	*5*	*Min Li*

Due to the current social distancing problem, we cannot get everyone's signature. All participation proof can be obtained from group leader.

The ***Project Leader*** *for this Group is: Thang Nguyen*

Signatures of Team/Project Leader: Thang Nguyen
Team Number/Name (if any) _________________

Note: *Team leader please briefly describe below if any group member(s) is/are not participating properly to justify learning and workload participation in the located task(s).*

____

____

____

____

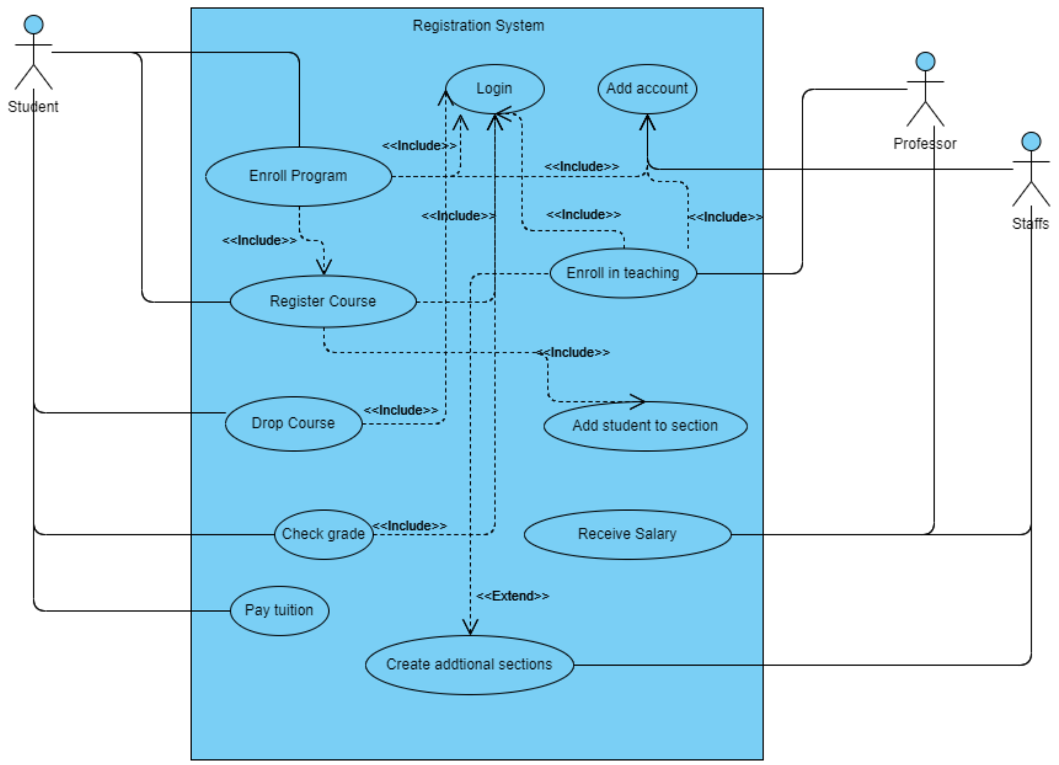
____

____

CST2234 – Final Exam

Algonquin College

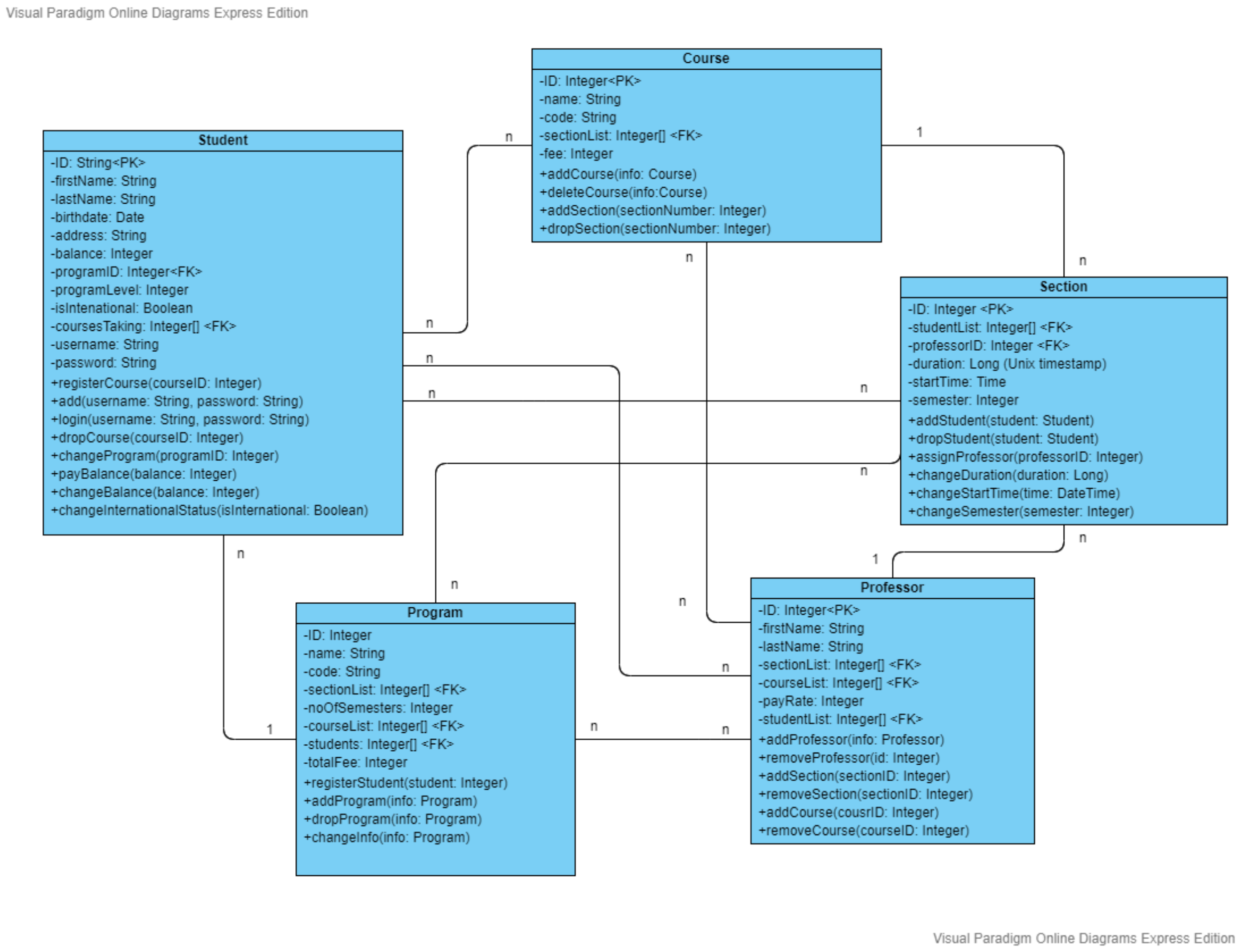
1. Use Case Diagram



Explaining the purpose of this diagram

Our use case diagram above shows the details of how students, professors, and staffs can interact with the Algonquin College registration system. Students can participate in many expected activities such as enrolling in programs, dropping courses from their program, checking their grades for the semesters, and paying tuition fees calculated from the courses they have taken. Each activity by the students will be carefully recorded and written in the database. Professors also benefit from the registration system by providing teaching enrollment information, and after that, the registration system will automatically handle the section assigning and student assigning. Staffs can create additional sections if they see that there are too many students in one section. All the required operations in the registration require all actors to login before they can continue with their tasks. **The purpose of this use case diagram is to outline all the possible activities of the registration system and map the activities together with their corresponding actors to provide better context information when building the real system.**

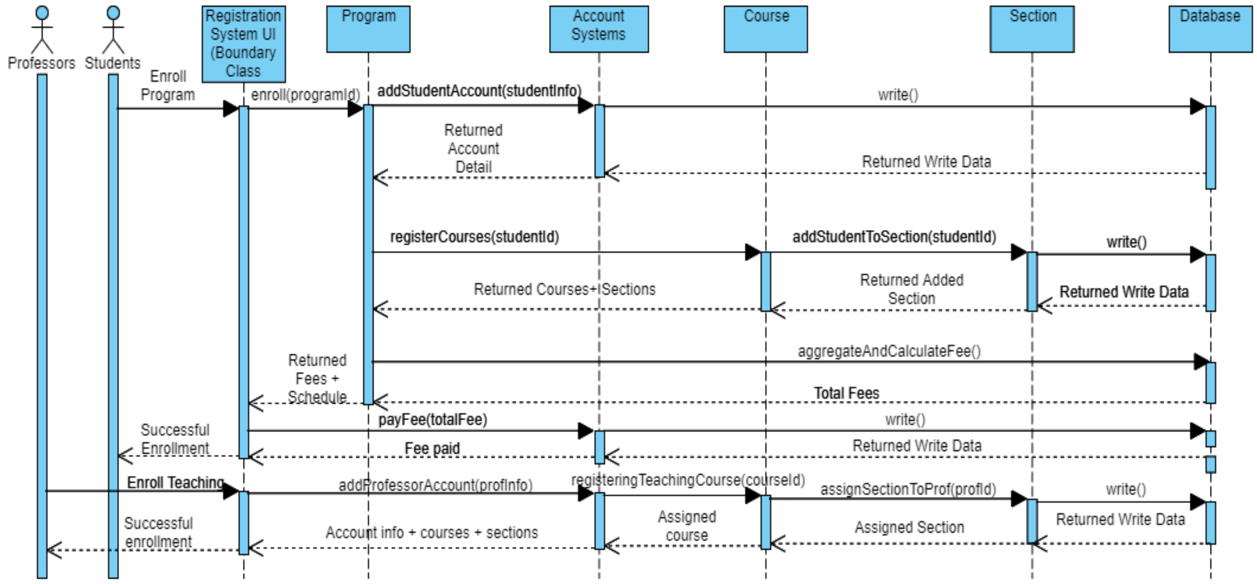
2. Class Diagram


Visual Paradigm Online Diagrams Express Edition
Course
-ID: Integer<PK>
-name: String
-code: String
-sectionList: Integer[] <FK>
-fee: Integer
+addCourse(info: Course)
+deleteCourse(info:Course)
+addSection(sectionNumber: Integer)
+dropSection(sectionNumber: Integer)
Student
-ID: String<PK>
-firstName: String
-lastName: String
-birthdate: Date
-address: String
-balance: Integer
-programID: Integer<FK>
-programLevel: Integer
-isIntenational: Boolean
-coursesTaking: Integer[] <FK>
-username: String
-password: String
+registerCourse(courseID: Integer)
+add(username: String, password: String)
+login(username: String, password: String)
+dropCourse(courseID: Integer)
+changeProgram(programID: Integer)
+payBalance(balance: Integer)
+changeBalance(balance: Integer)
+changeInternationalStatus(isInternational: Boolean)
Section
-ID: Integer <PK>
-studentList: Integer[] <FK>
-professorID: Integer <FK>
-duration: Long (Unix timestamp)
-startTime: Time
-semester: Integer
+addStudent(student: Student)
+dropStudent(student: Student)
+assignProfessor(professorID: Integer)
+changeDuration(duration: Long)
+changeStartTime(time: DateTime)
+changeSemester(semester: Integer)
Program
-ID: Integer
-name: String
-code: String
-sectionList: Integer[] <FK>
-noOfSemesters: Integer
-courseList: Integer[] <FK>
-students: Integer[] <FK>
-totalFee: Integer
+registerStudent(student: Integer)
+addProgram(info: Program)
+dropProgram(info: Program)
+changeInfo(info: Program)
Professor
-ID: Integer<PK>
-firstName: String
-lastName: String
-sectionList: Integer[] <FK>
-courseList: Integer[] <FK>
-payRate: Integer
-studentList: Integer[] <FK>
+addProfessor(info: Professor)
+removeProfessor(id: Integer)
+addSection(sectionID: Integer)
+removeSection(sectionID: Integer)
+addCourse(cousrID: Integer)
+removeCourse(courseID: Integer)
n
1
n
n
n
n
n
n
n
n
n
1
n
n
n
1
n
n
Visual Paradigm Online Diagrams Express Edition

Explaining the purpose of this diagram

The class diagram above shows the relationships between the entities inside Algonquin College registration system. There can be additional entities that are involved in the registration process, but for the sake of brevity, we decide to only mention five key entities that are present in the system. The student can only belong to one program, and once he is part of the program, he can decide to take as many courses as he wants. Each course can include many sections with one professor taking care of that section. The professor can choose to teach many courses, and he/she can also teach many sections that be long to the same course. Each entity has corresponding methods that help the staffs better manage of the registration system. **The purpose of this class diagram is to model the database entities that would exist inside the system and help us develop a system where business domain classes can strongly link to each other.**

3. Detailed Sequence Diagram



Explaining the purpose of this diagram

The above sequence diagram shows the detailed sequence steps of how students and professors can interact and enroll admission with the college registration system. *Please be noted* that there many additional actions which this diagram alone cannot cover including dropping courses or uploading grades. In order to keep our diagram concise, we have decided to go with 2 actions: student enrollment in their program of studying and professor enrollment in teaching. The business domain classes in this diagram include Program, Account Systems, Course, Section, and databases, while the boundary class includes the registration system UI. Students enrollment in a program allows our registration system to automatically add students to their required courses and sections, and the registration system will aggregate all the fees taken and return the fees to the students. Students can choose to pay the fees upright from the UI, and the enrollment process will return the successful response. Professors enrollment in teaching will also trigger the system to automatically assign the professors to their corresponding courses and sections. All changes will cause each sequence diagram class to have their own separate time/duration bar that handles the processes separately. **The purpose of this diagram is to show an example sequence of what is happening behind the system for each action caused by the user, and it help developers in creating sequential logic that can handle the registration process for students and faculty members.**

Part B

1) Ensure a smooth transition from the development team to the maintenance team. What factors should be considered and how to make this transition smooth and unquestionable.

There are many factors that should be considered in order to ensure a smooth project transition from the development team to maintenance team. **These factors include**:

+ The technical sophistication level of the project: This can include the measurement of the technical knowledge needed to develop and maintain this project. One has to asks themselves the questions of which type of people can be used to maintain this project.

+ The frequency or tendency of failure of the production project: Development team has to ask themselves the question of whether the project usually breakdown during regular uses or not? Identifying the most common problem/failure case of a project is very critical for future maintenance phase

+ Providing the resources that can be used for overall maintenance: Specifying the resources which the maintenance team can use to maintain the project can reduce the hassle in the future by cutting down all the resources they need to look at in order to fix the system.

In order to make the transition smooth and transparent, development teams have to come up with their own documentation that cover all the factors mentioned above. Documentation has to have a section that is specifically designed for future maintenance team, and it has to be concise without missing any of the important technical detail. If a project has a very deep level of technical sophistication, documentation can provide a link to a tutorial video by the development team to better cover the required knowledge to maintain the system. It is wise to remember that graphical images and directions always have better effects on learning than using textual knowledge. If a project tends to fail, identifying common problems in one of the documentation sections so maintainers can fast-track their diagnosing process so that they can provide fixing solution in a timely manner. It is also to critical to remember that maintenance team usually does not have enough depth of knowledge to fix the whole system, and therefore, providing a utility resources or tools that can help the maintainers diagnose the system quicker can make the whole project transition much more smooth and stress-free.

2) What are the risk factors which may affect or create hurdles during this transition ?

There are many risk factors that can affect hurdles during the transaction. Some of them include:

+ Underestimating the scope and depth of this project: When transitioning from the development team to maintenance team, project owners usually make the mistake of not fully grasping the technical depth of the project which leads to **miscalculating** the required resources to maintain the project. If a project has a very steep technical sophistication, providing little resources to the maintenance team can take a lot of time to the maintenance process should there are any problems. Sometimes situations may get worse, and the problems cannot be fixed at all because there are simply not enough knowledge or people resources to tackle them.

+ Providing missing resource credential information to the maintainers such as production database account usernames or passwords: This is a very common risk factor that is associated with the transition process. During development phase, it is very common to see developers forgetting write down all the necessary credential information that is required to maintain parts of the systems. Some third-party system providers such as Amazon Web Services or Google Cloud Platform that provide database system or storage system require username and password in order to view all the resources inside our systems. If the maintainers happen to not have access to any of those resources, troubleshooting and maintenance will take a lot of time due to no access to required information.

+ System failure because maintenance team is not aware of the interconnectivity between system components: When providing maintenance to the project, a maintainer has to be strongly aware of the overall effect of the component he/she is fixing on the overall system. If he/she does not fully grasp the overall architecture, the maintainer may accidentally break the whole system by providing fix to the wrong part of the project. Therefore, it is very critical for the maintenance team to be aware of the effect of the actions which they will perform on top of the system component.

3) What form(s) of testing should be done during the development phase to ensure successful maintainability ?

A project can only be easily maintained when the product satisfies all the requirements from the client. In order to reach the goal of providing future maintainers with a reliable and easy to fix product, a project has to follow some of the testing methods to ensure that it is easy to fix and maintain. This form of testing is commonly known as maintainability testing. Maintainability testing is “the process of testing the system’s ability to update, modify the application if required” [1].

In order to perform maintainability testing, the tester should “verify the development of the standards for structured programming, standards for database approach and standards for user interfaces”. All the inputs, processing, and outputs are verified regardless of how cohesion they are implemented. Testers also need to verify the reusability of features by “assigning proper parameter with necessary conditions for programs”. Furthermore, the system should be “properly distributed and algorithms should be optimized” [1].

According to Priya, maintainability testing has four sub testing categories that we need to follow during the development phase to ensure successful maintainability in the future. These four categories are [1]:

+ Corrective maintenance testing: Corrective maintenance testing measures how fast a maintainer can diagnose and fix the problems

+ Perfective maintenance testing: Perfective maintenance testing measures the speed of a maintainer in how he or she can make the effort for enhancements.

+ Adaptive maintenance testing: Adaptive maintenance testing measures the resilient ability of how a maintainer can make the adaptations required by the system in according to the changing environments.

+ Preventive maintenance testing: Preventive maintenance testing is conducted to reassure the cost of future maintenance.

References

1) T. Mishra and M. Polsani, “MAINTAINABILITY TESTING,” *H2kinfosys Blog*, 03-Mar-2019. [Online]. Available: https://www.h2kinfosys.com/blog/maintainability-testing/. [Accessed: 19-Apr-2020].