Course Management System DOCUMENTATION

1. INTRODUCTION

The requirements of the CMS are based on the work of [Baniassad, Clarke 04] and [Clarke, Baniassad 05]:

R1. Students can enroll in individual courses.
R2. Students can drop individual courses.
R3. Each student enrolling in an individual course must be registered.
R4. Each dropped course must be registered.
R5. Teachers can discharge students from an individual course.
R6. Each student discharge must be registered.
R7. When students are discharged from a course, they must be labeled as special.
R8. Teachers can grade student coursework.
R9. Each student grade must be registered.

We identified two types of actors–students and teachers–and three UCs: Enroll Student, Delete Student Enrollment, and Grade Student. Also, requirements R3, R4, R6, R7, and R9 were found to be cross-sectional with respect to the other requirements, since they all imply registering the resulting activity in a follow-up file. These requirements are encapsulated in an aspect identified as UC Register Follow-Up, and it is included in the rest of the UCs with stereotype <<trigger>>.

The project was codified using Cal-4700 language. Cal-4700 projects work in a single folder, in which you should find the main file, “the noodle”, which contains all language definitions and routines. The source code files in the folder must have no extensions, and when compiled as a whole, they allow to separate the solution from the source code.
When compiling any of the programs in the folder, an executable file will be generated, whose name will be equal to the name of the project folder itself. In this case the project folder is called “cm system” and contains:

- File “enroll”, contains the routines that allow to register the enrollment of a student in the text file Students.txt.
- File “unenroll”, contains the routines that allow to remove a student from a course. According to the requirements, the student is marked as a drop, in this case the letter “D” is prefixed inside the text file Students.txt. The data diagram provided a reference point for defining the structure of the persistent data. However, when working with text files, it was decided to handle each enrolled course in a different way, independently, facilitating

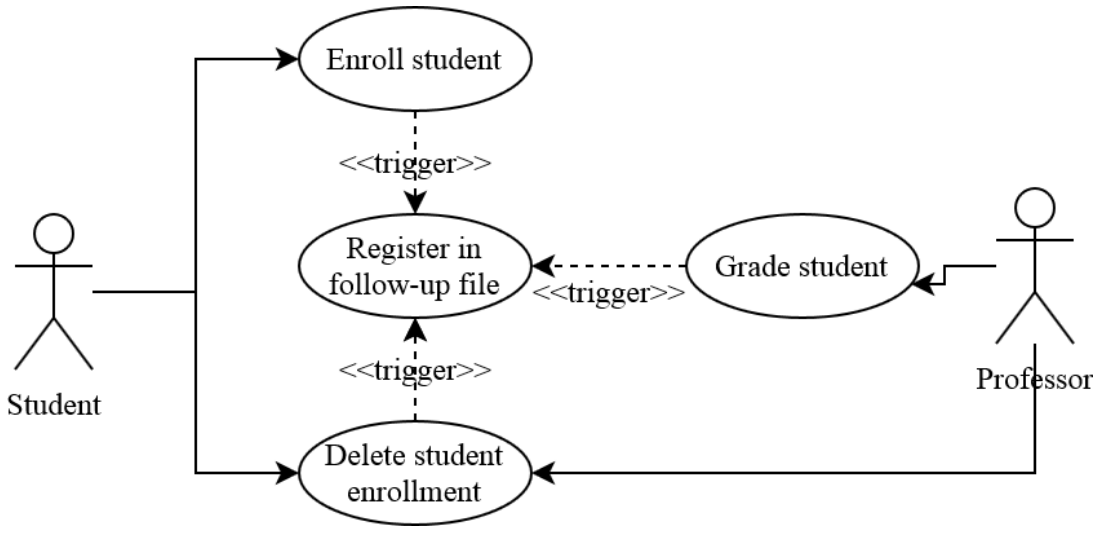
the grading of students, the access and reading from the outside. It is therefore assumed that there is a process that divides the enrollments into different text files (omitting the students marked with “D”), according to the course.

- File “grade”, contains the routines that allow to record the grade of a student.
- “Main” file, contains the routines for the start and the end of the program (*To run:*, *start up.*, *shut down.*), which are necessary to create the executable program and call the rest of the routines of each UC.
- “aspectLog” file contains the routines corresponding to the system tracking.
- The persistent data text files are also located in this folder, subfolder “files” (Students.txt, Log.txt, as well as the files corresponding to each course of final enrollees, 1.txt, 2.txt, 3.txt).

Following sections presents the corresponding Analysis and Design diagrams according to the Naturalistic Method proposed by Hernández-González L.A and Juárez-Martínez Ulises.

2. OVERVIEW DIAGRAMS

2.1. Use Case Diagram



2.2. Atomic ideas

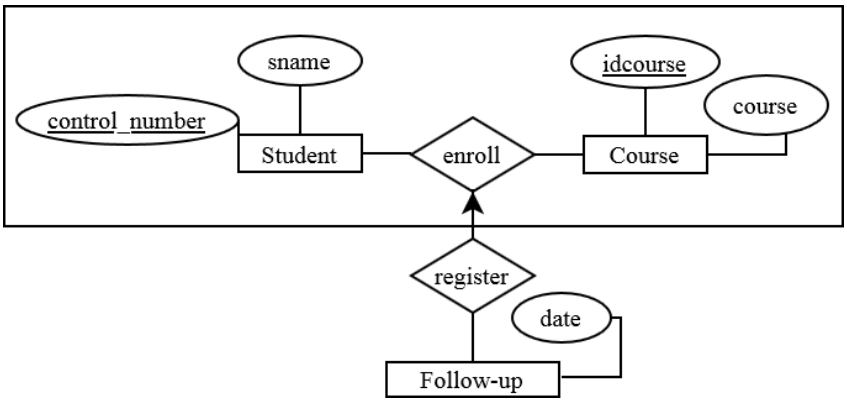
Student
control_number : Number sname : String

Course
course_number : Number cname : String

Enrollments
control_number : Number course_number: Number grade: Number mark : String

Follow-up
control_number : Number course_number: Number date: String trans : String

2.3. E-R diagram



Hereafter, for every Use Case, description, prototype and tabulation of ideas are presented, as well as Data Flow Diagram (DFD) and Structure Diagram.

3. USE CASE ENROLL STUDENT

3.1. Description and prototype

ID:	UC01
Name:	Enroll student
Author(s):	Lizbeth A. Hernández González
Date created:	Apr 15, 2021
Date updated:	Apr 30, 2021
Actor(s):	Student
Description:	The student selects the desired course(s) from the provided list
Normal flow:	1. System deploys course list. 2. System requests student ID number. 3. Student enters it. 4. System requests student course ID number. 5. Student enters it. 6. System asks: Do you wish to enroll in another course? 6.1 Yes: Return to step 4 6.1 No: End UC 7. System prints out record of enrolled courses.
Alternative flows:	
Exceptions:	E.1 Course full 1. System displays "Course with no places available". 2. Return to step 6
Postconditions:	Record the Student in the log file, along with the enrolled courses. Indicate the transaction performed.
Entries:	The student's course number and control number.
Outputs:	Confirmation message.
Include relation:	
Extend relation:	
Launch relation: (<<trigger>>)	Record follow-up
Priority:	High

Course enrollment

3. NodeJS
2. JavaScript
1. MongoDB
0. Exit

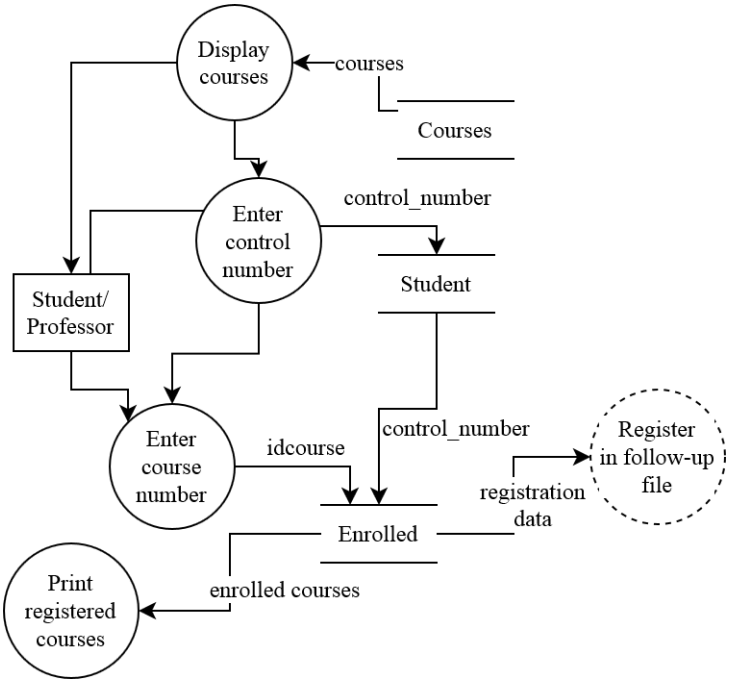
Enter student ID number: 13011074
Select a course: 1
Successful registration!

Do you wish to enroll in another course? 1-Yes 0-No
0
[student ID number: 13011074 course:1]

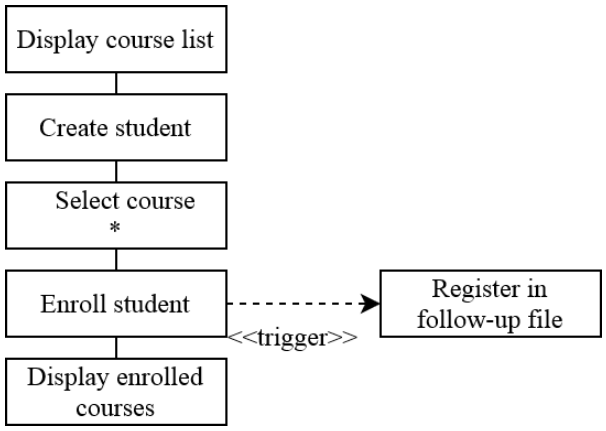
3.2. Tabulation of ideas

No.	Statement (action)	Command (phrase)	Condition	Agent	Anaphoric Reference	Direct object *plural nouns, only	Destination (optional)
1	Display			System		Course	
2	Request			System		student ID number	
3	Capture			Student	student ID number		
4	Request			System		course ID number	
5	Capture			Student	course ID number		
6			Ask	System		Course	
6.1		Yes: Execute		System		Step 4	
6.2		No: Exit		System			
7	Print			System	Enrolled courses		

3.3. DFD



3.4. Structure diagram



4. USE CASE DELETE STUDENT ENROLLMENT

4.1. Description and prototype

ID:	UC02
Name:	Delete student enrollment
Author(s):	Lizbeth A. Hernández González
Date created:	Nov 10, 2021

Date updated:	
Actor(s):	Student, Professor
Description:	The Student or the Professor may delete a student's registration, by means of the student's control number and the corresponding course number.
Normal flow:	1. The system displays the list of courses. 2. The system requests the student's control number. 3. Student or Professor enters it. 4. The system requests the number of the course to be cancelled. 5. Student or Professor enters it. 6. System marks the student as a drop of the specified course. 7. System asks Do you want to drop another course? 7.1 Yes: go back to step 1. 7.2 No: terminates. 8. System displays number of courses dropped.
Alternative flows:	
Exceptions:	E.1 Student does not exist 1. System displays “Student does not exist.” 2. Return to step 1. E2. Course number does not exist 1. System displays "Course number does not exist.” 2. Return to step 5.
Postconditions:	Record the Student's withdrawal in the log file, using his or her control number and the number of each course withdrawn. Indicate the transaction performed.
Entries:	The student's control number as well as the number of each course to be dropped.
Outputs:	List of unsubscribed courses.
Include relation:	
Extend relation:	
Launch relation: (<<trigger>>)	Record follow-up
Priority:	High

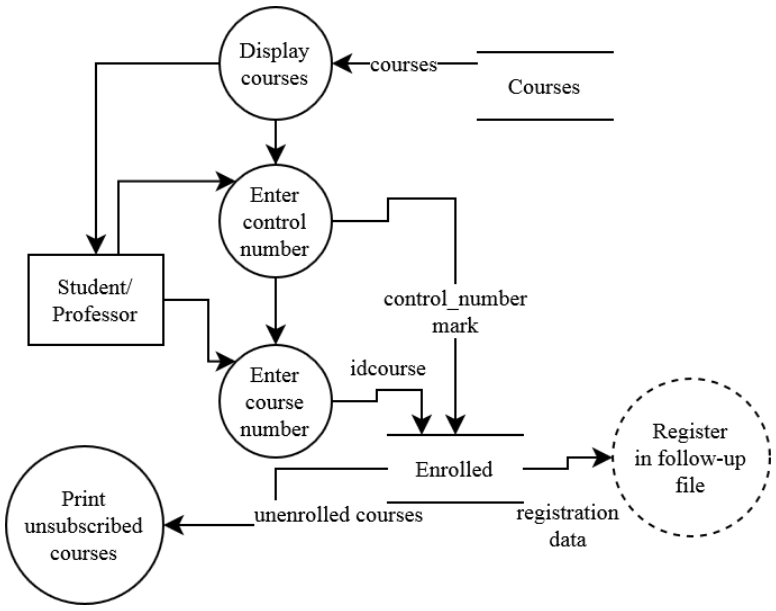
Course unenrollment

```
3. NodeJS
2. JavaScript
1. MongoDB
0. Exit

Enter student ID number:  13011074
Select a course: 1
Student has been unsuscribe!

Do you wish to unsuscribe another student? 1-Yes 0-No
0
```

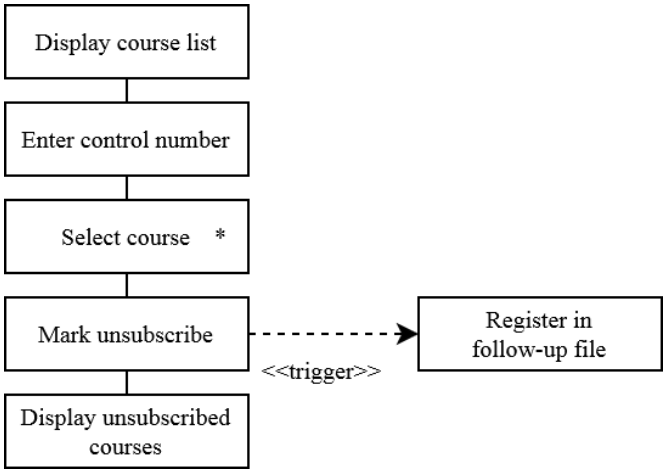
4.2. Tabulation of ideas

No.	Statement (action)	Command (phrase)	Condition	Agent	Anaphoric Reference	Direct object *plural nouns, only	Destination (optional)
1	Display			System		Course*	
2	Request			System		Control_number	
3	Capture			Professor	Control_number		
4	Request			System		Course_number	
5	Capture			Student/ Professor	Course_number		
6	Mark(saving)			System		Student	Students
7			Ask	System		Course	
7.1		Yes: Execute		System		Step 1	
7.2		No: Exit		System			
8	Print			System	Courses* cancelled		

4.3. DFD



4.4. Structure diagram



5. USE CASE GRADE STUDENT

5.1. Description and prototype

ID:	UC03
Name:	Grade student
Author(s):	Lizbeth A. Hernández González
Date created:	Nov 16, 2021
Date updated:	
Actor(s):	Professor
Description:	The Professor selects a course to record grades for each student.
Normal flow:	1. The system displays the list of courses. 2. System requests course number. 3. Professor enters it. 4. System asks Do you want to grade the whole group [1] or one student [2]? 5. Option 1: 5.1 System displays one by one control number (of students). 5.2 Teacher records grade for each student. 6. Option 2: 6.1 System prompts for student control number. 6.2 Professor records grade. 7. System saves grades.
Alternative flows:	
Exceptions:	E.1 Student does not exist 1. System displays "Student does not exist.” 2. Return to step 1.

	E2. Course number does not exist 1. System displays "Course number does not exist". 2. Return to step 5.
Postconditions:	Record the grade of the student(s) in the log file, using their control number, course number and type of transaction performed.
Entries:	The course number, if the teacher wants to grade only one student, he/she must enter his/her control number.
Outputs:	Confirmation message.
Include relation:	
Extend relation:	
Launch relation: (<<trigger>>)	Record follow-up
Priority:	High

Grading students

3. NodeJS
2. JavaScript
1. MongoDB
0. Exit

Select a course: 1
Grade everyone [1], Select a student [2]: 1
Enter grade for 43257: 8.3
Enter grade for 89542: 9.4
Enter grade for 45629: 8.3
Enter grade for 34261: 5.6

Grades registered!

Grading students

3. NodeJS
2. JavaScript
1. MongoDB
0. Exit

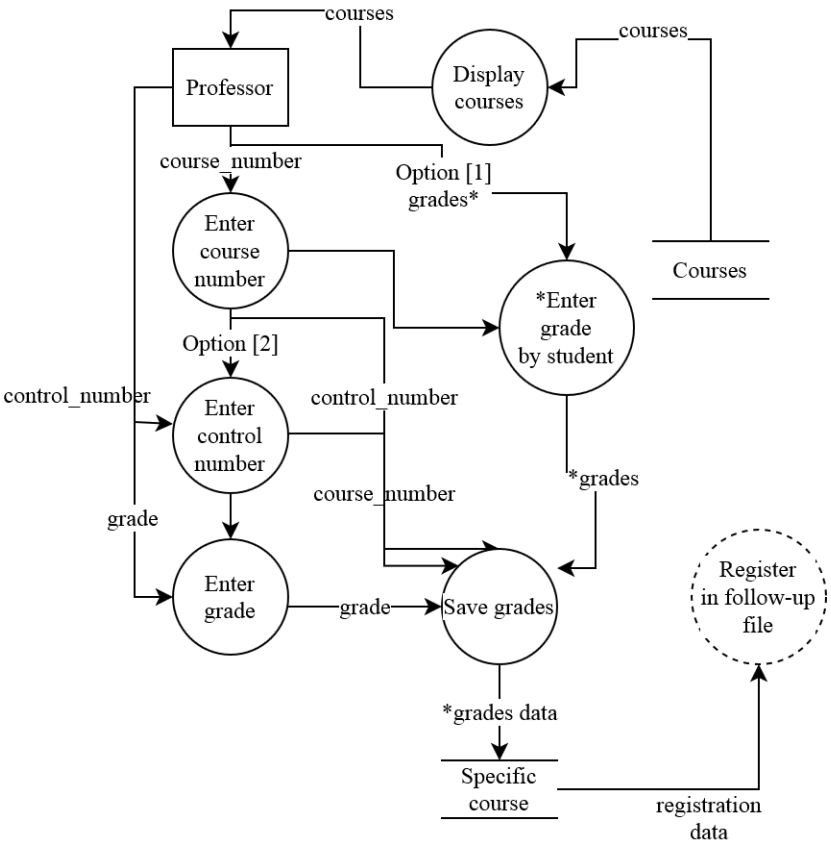
Select a course: 1
Grade everyone [1], Select a student [2]: 2
Enter the control number: 45629
Enter the grade: 8.3

Grade registered!

5.2. Tabulation of ideas

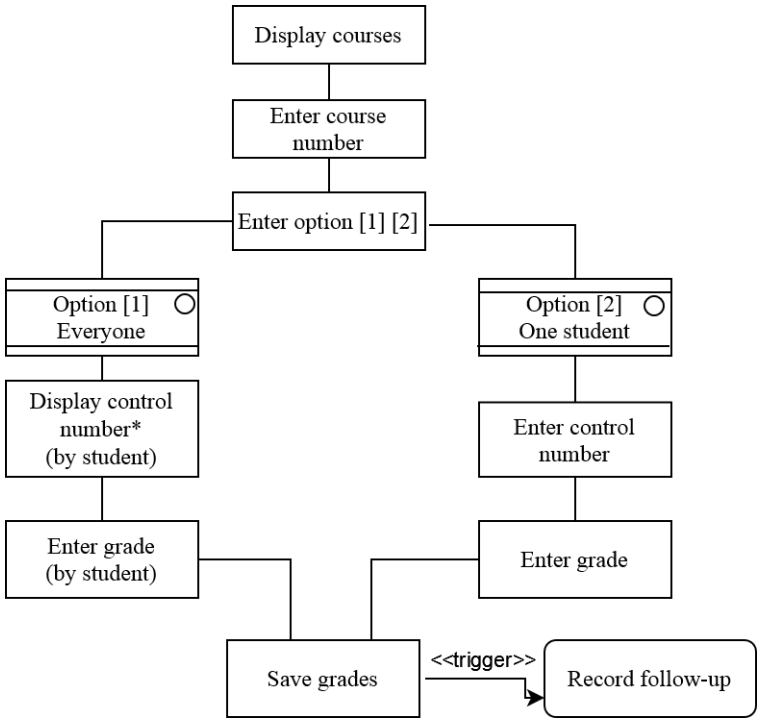
No.	Statement (action)	Command (phrase)	Condition	Agent	Anaphoric Reference	Direct object *plural nouns, only	Destination (optional)
1	Display			System		Course*	
2	Request			System		course_number	
3	Capture			Professor	course_number		
4			Ask	System			
5			Option 1				
5.1	Display			System		control_number by Student *	
5.2	Capture			Professor		Grade by Student *	
6			Option 2				
6.1	Request			System		control_number	
6.2	Capture			Professor		Grade	
7	Save			System		Grades	Specific course file

5.3. DFD



Note: "*" means that the user must enter several data, in this case, several grades, one for each student.

5.4. Structure diagram



6. ASPECT REGISTER IN FOLLOW-UP FILE

6.1. Description and prototype

There is no interactive interface for an aspect, the triggering is done through the Use Case who launch (<<trigger>>) the aspect, directly.

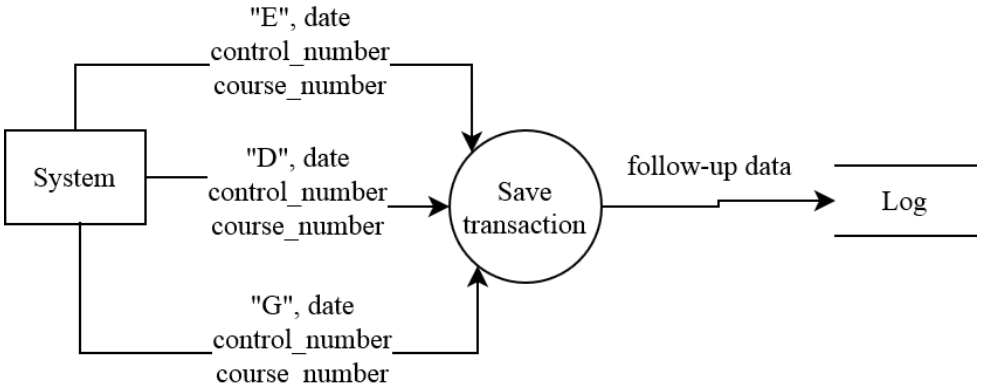
ID:	AS01
Name:	Register in follow-up file
Author(s):	Lizbeth A. Hernández González
Date created:	Nov 17, 2021
Date updated:	
Description:	The aspect records in a log file the transactions performed: Enroll, Delete enrollment, Grade.
Normal flow:	1. Enroll transaction: 1.1. System stores "E", date, control number, course number. 2. Transaction Delete enrollment: 2.1. System saves "D", date, control number, course number.

	3. Grade Transaction: 3.1 System saves "G", date, control number, course number.
Entries:	Transaction type, date, course number and student control number.
Priority:	High

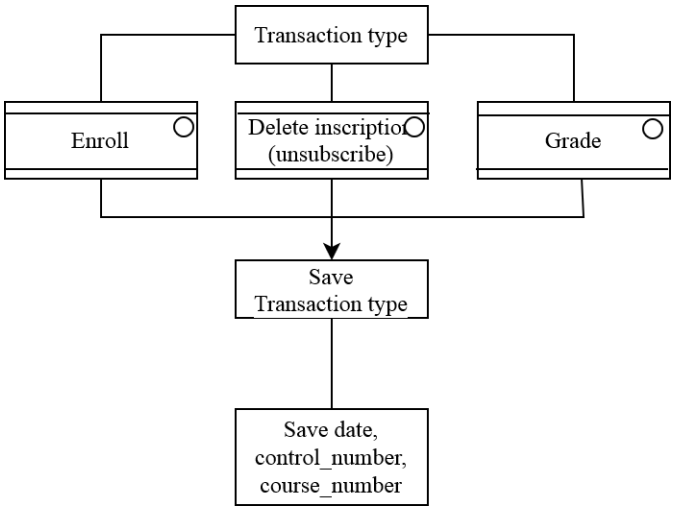
6.2. Tabulation of ideas

No.	Statement (action)	Command (phrase)	Condition	Agent	Anaphoric Reference	Direct object *plural nouns, only	Destination (optional)
1			Option E: enroll				
1.1	Save			System		“E”,date,control_number, course_number.	Log
2			Option D: delete enrollment				
2.1	Save			System		“D”,date,control_number, course_number.	Log
3			Option G: grade students				
3.1	Save			System		“G”,date,control_number, course_number.	Log

6.3. DFD



6.4. Structure diagram



References

[Baniassad, Clarke 04] Baniassad, E., Clarke, S.: “Theme: An Approach for Aspect-Oriented Analysis and Design”; In Proceedings. 26th International Conference on Software Engineering, (2004), pages 158–167. doi: 10.1109/ICSE.2004.1317438. ISSN: 0270-5257.

[Clarke, Baniassad 05] Clarke, S., Baniassad, E.: “Aspect-Oriented Analysis and Design, The Theme Approach”; Object Technology. Addison-Wesley, first edition ISBN 0-321-24674-8. (2005)