**A Software Design Specification For Dynamic Course Planner**

by Daniel Castro <DanielCastro@letu.edu>

DK <DkKim@letu.edu>

Ethan Sikes <EthanSikes@letu.edu>

COSC 3403

Software Engineering I

2/24/2017

Copyright © 1994-1997 by Bradford D. Appleton

*Permission is hereby granted to make and distribute verbatim copies of this document provided the copyright notice and this permission notice are preserved on all copies*.

**Table of Contents**

1. Introduction.. 1

1.1. Document Purpose.. 1

1.2. Product Scope.. 1

1.3. Intended Audience and Document Overview.. 1

1.4. Definitions, Acronyms and Abbreviations.. 1

1.5. References and Acknowledgments.. 2

2. Design Considerations.. 3

2.1. Assumptions and Dependencies. 3

2.2. General Constraints. 3

2.3. Goals and Guidelines. 3

3. Architectural Strategies. 4

4. System Architecture.. 6

4.1. Subsystem Architecture. 6

5. Policies and Tactics.. 8

5.1. IDE.. 9

5.2. Coding guidelines and conventions. 10

6. Detailed System Design.. 10

6.1. Classification. 10

6.2. Definition 10

6.3. Responsibilities. 10

6.4. Constraints. 10

6.5. Composition.. 10

6.6. Uses/Interactions. 11

6.7. Resources. 11

6.8. Processing. 11

6.9. Interface/Exports. 11

6.10. Detailed Subsystem Design.. 12

7. Glossary.. 13

# **1.** **Introduction**

The dynamic course planner is an academic career planner that allows a student to dynamically plan all four or more years of their classes in one place. It allows a student to select their major and manually add courses to the planner. They will be able to account for classes taken over the summer or transferred from other institutions. Course plans can be exported, imported, and converted to a pdf format.

## **1.1 Document Purpose**

This document outlines the design specifications for the Dynamic Course Planner software engineering project. Its purpose is to give the implementation team a guide toward a completed product. These design specifications continue to fill in some gray areas left by the SRS, provide a clearer understanding of the end product, and describe how the project should be executed.

## **1.2 Product Scope**

The purpose of the Dynamic Course Planner software product is to supplement and guide students in their semesterly course planning. The product will help to encourage interactive planning between the student and the course options. The software is to diagram the semesters of classes each student will take in their degree program.

## **1.3 Intended Audience and Document Overview**

This particular design document shall be informative to the designers, developers, project creator, and manager (Dr. Baas) as to further details and guidance of the Dynamic Course Planner from the baby stages during prototype development. The “Design Considerations” section will set the framework for the designs that follow, by stating the project expectations, guidelines, and constraints. The “Architectural Strategies” will be discuss the policies pertaining to each part of the development of the end product. “System Architecture” covers the general design of the product and how subsystems relate to each other. “Policies and Tactics” details the coding practices and guidelines. “Detailed System Design” further specifies how the product is to be designed and structured.

## **1.4 Definitions, Acronyms and Abbreviations**

**DCP:** Dynamic Course Planner

**GPA:** Grade point average

**IEEE:** Institute of Electrical and Electronics Engineers

**Project Creator:** Corey Naas

**Project Manager:** Dr. Baas

**Student:** Primary end user/client

## **1.5 References and Acknowledgments**

This document is based the SRS previously created for this project and is further clarified by the SEI textbook.

“Software Requirements Specification for Dynamic Course Planner” prepared by Daniel Castro, Baughn Welch, and Nathan Kleoppel. February 6, 2017

I. Sommerville, *Software Engineering (9th edition)*, 9th ed. Boston: Addison-Wesley Educational Publishers, 2010.

# **2.** **Design Considerations**

This section describes many of the issues which need to be addressed or resolved before attempting to devise a complete design solution.

## **2.1.** **Assumptions and Dependencies**

* Student knows what meets the requirements of a specified elective credit.
* The student is taking university approved courses and enrolled in a degree that is university sanctioned.
* Student knows what a reasonable course load is for a single semester.
* Student is knowledgeable about basic user interfaces in general without trouble dragging classes to semesters.
* Software will be run on the Windows platform.
* The management of related files will not be interfered with by the user or other parties.

## **2.2.** **General Constraints**

* Developed on and for Windows OS
* Programmed in C#
* Developed using Visual Studio and the .NET Framework
* Final product has no reliance on outside databases or non-OS software
* Has clear and helpful user interface
* Course plans can be safely saved and retrieved
* No significant performance requirements
* The product shall protect the privacy of student grades with necessary consideration.
* Student information and entries shall not be transferred if program is shared between user systems.
* Must be thoroughly tested to ensure fulfillment of the SRS

## **2.3.** **Goals and Guidelines**

* The KISS principle ("Keep it simple stupid!")
* The interface and code will be clear, concise, and consistent
* Reliability - The product shall be free of bugs
* Consistency - Product format will remain constant from subsystem to subsystem
* Adaptability - Software is able to be used by varying institutions and users
* Usability - Interfaces are simple and straightforward

**3.** **Architectural Strategies**

1. Programming Language: The following program (DCP) should be written with Visual Studio C#, which are great for making well represented graphical user interface.
2. Database: The database of this program should preferably imported from the course catalog itself (in this current stage, it should be implemented manually; however, if it is available, this should be implemented from the LetNet database).
3. Library: Use the given libraries for the graphical user interface for C# using Visual Studio.
4. Reuse: No relevant programs known for reuse.
5. Future Plans for Enhancement: Possibly, import the degree audit of the user when the user provides the program with his or her student ID. Then, with the imported profile from the degree audit of that student, the user will be able to effectively and efficiently plan the course schedules for the entire college career. When this program gets even more enhanced and developed, the program will be used to actually register for classes.
6. User Interface Paradigms:
   1. User input: Dragging the courses into the semester boxes.
   2. System output: Course plan in pdf format.
7. Hardware and/or Software Interface Paradigms:
   1. Refer to Section 3.1 on software requirements specification document for this program.
8. Error Detection and Recovery
   1. Validation of course name, course code, and requirements:
      1. If the user enters in invalid characters such as emojis or special characters, the program should analyze such input as an error and not allow the user to actually enter in any invalid characters in the course text box.
      2. If the user drags in the course into a semester box, but the prerequisite course of that course is either not in the previous semester boxes of the semester boxes with that course or is in the semester boxes after the semester boxes that the course is in, detect it as an error and change the font color of the wrongly placed prerequisite course to red.
9. Memory Management Policies:

No memory management is needed as the program will be using very insignificant amount of memory (expected memory usage: <300KB).

1. External Databases and/or Data Storage Management and Persistence:

There is no external databases to manage, as the course data will likely be entered into the program manually by the user (at this stage; if this program goes through further development and gets the features as mentioned in No. 5, there need be network/cloud data management as the course catalog and the degree audit will be imported from the school server.

1. Distributed Data or Control over a Network:

There will be no distributed data or control over a network in this version of the program. However, if the later versions of the program implements the database from the LetNet, it will be importing and using the degree audit data and the course catalog database.

**4.** **System Architecture**

# 

As seen from the above UML diagram of this software, the system is divided in 7 different classes. First, the main window is the class for the form (or the system GUI) to provide the user an environment for inputs and outputs to this program. Briefly, this main window class will have the tree-viewer on the left to show the list of courses and the table with cells to represent different semesters, depending on how many semesters that the user has.

From such partitions of the main window, there will be the SemesterData class for the data of each specific semester, which will be pertaining to the cell on the table on the main window. In this SemesterData class, there will be a ClassData class which will be used to store the information about the courses that are in that specific semester. Therefore, each SemesterData class for each cell will most likely have several ClassData classes that provide the information about the courses that are part of that specific semester.

The AddClassWindow and the AddSemester classes are both functionally similar. The AddClassWindow class will be used to manually add the courses to the tree-viewer list on the left of the main window. It will be providing the user a dialog box to do the task. There will be few elements on the class to help the user with adding the courses. Basically, there will be a text box for the class name and the options (radio buttons) to indicate the availability of the course. The AddSemester class will be used to add more semester “cells” along with the SemesterData classes pertaining to the cells, in case the user requires more than the default 8 semester cells on the main window. This will also have a dialogue box to aid the user with manually adding the semester to the main window; in this dialogue box, there will be a drop-down menu to allow the user to choose and specify the season of that semester. There will be no text boxes for the user to specify the name or the number of that semester as all of the semesters will be automatically numbered.

Finally, there will be the File Handling class and the Import class for handling files within the software and possibly importing the data files for the software. The File Handling class and the Import class will be used interactively to each other, as the loadFile() method on the File Handling class will call the Import class run its functions. While the File Handling class will be interacting with the local data sources “outside” the software, the Import class will be dealing with the data inside the software, meaning that it will mainly used to read in the data from the loaded file and implement the database from the loaded file into the software.

As this software is a fairly simple software (without further implementation of the network databases as mentioned in the section 3.5), this software does not have any other complex subsystem that needs to specified in terms of its design and architecture.

**5.** **Policies and Tactics**

# **5.1 IDE**

Visual Studio IDE should be used when implementing the product. Alternatively, MonoDevelop or SharpDevelop could be used for developing and implementing, however most programmers would most likely feel comfortable working with Visual Studio. In addition, Visual Studio has an easy to use GUI builder interface. Therefore, using the windows form designer is highly recommend.

**5.2 Coding guidelines and conventions**

**5.2.1 Layout Conventions**

Use the default Code Editor settings (Smart indenting, four-character indents, and tabs saved as spaces). Write only one statement per line and only one declaration per line. If lines are not indented automatically, then indent them one tab or four spaces. Use parentheses to make clauses in an expression apparent, as shown below:

If(//condition)

{

//expression

}

**5.2.2 Commenting Conventions**

Place the comment on a separate line and begin comment text with an uppercase letter. If describing a line or function, comment above it. End comment text with a period and insert one space between the comment delimiter (//) and the comment text.

**5.2.3 Implicitly typed local variables**

Only use implicit typing for local variables when the type of the variable is obvious. For example:

var num = 27;

Avoid the use of var data type as much as possible.

**6.** **Detailed System Design**

## **6.1.** **Responsibilities**

**6.1.1 Drag and Drop**

Drag and drop means to select a class name from the class library, which is listed in the treeviewer, and drop it onto a semester cell in the table. When the event occurs then the properties of the class selected will be copied onto the cell.

**6.1.2 Import**

The import component is responsible for organizing the user’s profile course planner text file. Therefore, the user can export and edit or view the profile.

**6.1.3 Export**

The export component is responsible for reading the profile text document and displaying it onto the application.

## **6.2.** **Interactions**

**6.2.1 Import**

The import component should be called by the file handling “load file” method and the file handling component should be called by the import button handler method in the main window class. So, the loadfile method will read a string and pass the to the import class which then it would organizes it and then later display the properties in the course planner program.

**6.2.2 Export**

The export component should be called by the load file method in the file handling class. Which the savefile method, as you can guess, will be called by the export button handler method in the window class. So, the responsibility of the export component is not necessarily making the profile text file, but to get the required elements to help create the file.

## **6.3.** **Exports**

**6.3.1 Propose**

This section will only explain how to design the profile text document.

**6.3.2 Profile document**

The program should split components into its own tag. Some of those components are:

1. Profile - the name and properties of the profile
2. Class Library - a list of all the classes in the treeviewer
3. Semesters - a list of all the semesters that were in the table

The program make a text document like so:

[P]

N=“Steve Rogers”

#S=“1”

#CL=“3”

CG=1.5

[CL]

Names=Computer Science I;TheCoolestClass 101;PencilSharpener 102

C#=COSC1923;COOL1231;LAME2006

O=B,Fall,B

C3=LAME1006

[S1]

#C=2

O=Fall

C#=COSC1923;LAME0006

CG=D;A

TG=1.4

The first line is the begin of a new profile, and the line after that is the name of the profile. The third and fourth line is the number of semesters and classes (that are in the class library) respectively. The final line in the below the profile tag is the career gpa.

Next, in the sixth line, is the beginning of a new class library. The following four line are the properties of the class library: names of classes, class codes, the class term offered, the class prerequisites. The ninth line, is basically, when the class is offered. In this example “TheCoolestClass 101” is offered only in the fall while the other two class are offered during both semesters (which is represented by the ‘B’ Character). The last line in the class library tag is the prerequisites for a certain class. In this example, since it is “C3” it is referring to the third class in the class library, thus the PencilSharpener 102 has a prerequisites class coded LAME1006.

Finally, in the eleventh line, is the beginning of the first semester. The next line is the number of classes in the semester. The following line is the semester term, and the next line is the class codes that are in the semester. The last two lines are the grade in the class and the total gpa respectively. If the user has not enter a grade for a class then the ‘$’ symbol should be used as a place holder.