# Software Architecture / SOEN 6471

# Project Name : A+ Student Planner

## Dr. Peter C Rigby

## Winter 2013

## Document Name : M4


| | |
|---|---|
| **Jamileh Mohammadkhani Ghiyasvand** | **Student ID : 5990351** |
| **Bahareh Mohammadpourbarghi** | **Student ID : 5894336** |
| **Shahla Noori** | **Student ID : 5972671** |
| **Robert Saliba** | **Student ID : 6412033** |
| **Harcharan Singh Pabla** | **Student ID : 6547702** |

## Project Description

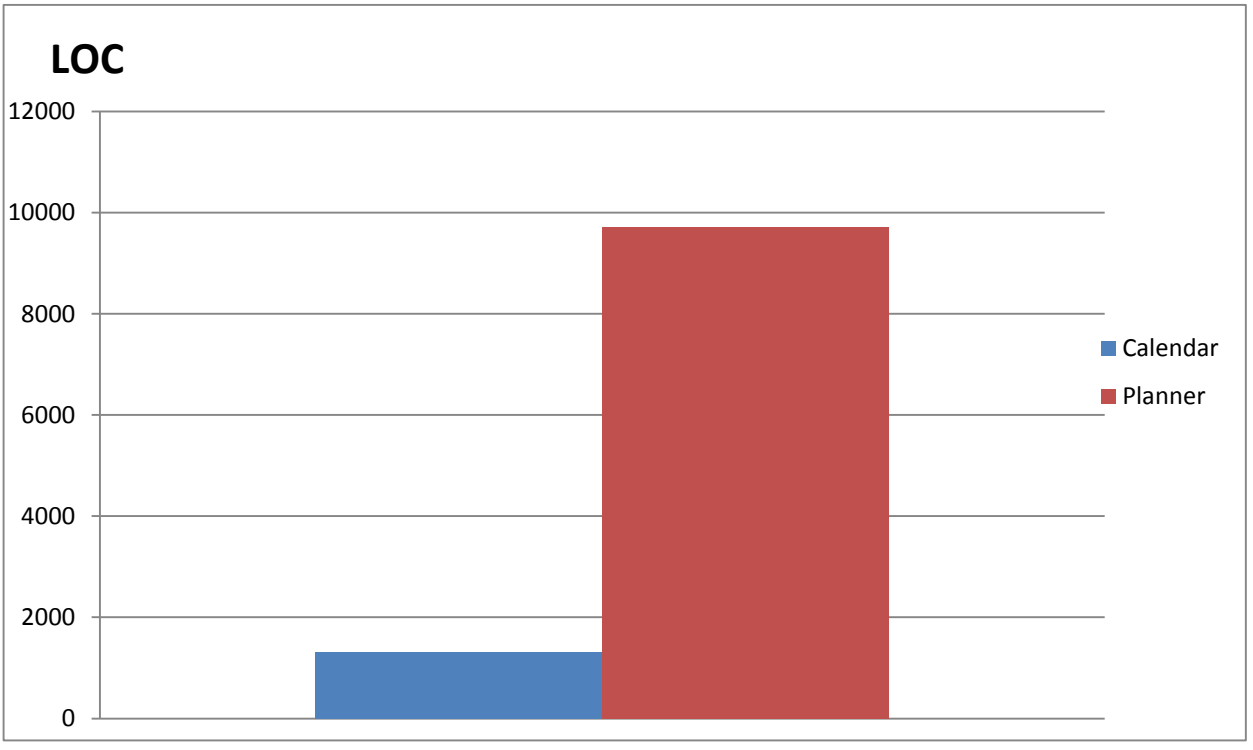
This application provides an easy use to keep track of student's assignments, events and current grades in all of his/her classes. Additionally, it is well integrated with Google Calendar and supports for sending email messages to professors within the program.

The version of this project is Beta and the last update is in 2011-08-31

## Motivation and Project Size and Scope

Based on the article published[1] in June 2011, small to medium size of a Java-based project would have a limitation of 170K SLOC. By considering the total lines of code of our application, the number of classes and their SLOCs and also the number of members in our group, we figure out that this application with 11K SLOC could be a term project and architecturally should be simple to understand.

This application contains 2 projects named Calendar and Planner, accumulatively having 11029 lines of code with a total of 108 csharp and resx files. The following chart illustrates the Line of code of the project.




[1] International Journal of Computer Science & Information Technology (IJCSIT), Vol 3, No 3, June 2011
STATISTICAL ANALYSIS ON SOFTWARE METRICS AFFECTING MODULARITY IN OPEN SOURCE SOFTWARE

## Group Members

**Jamileh Mohammadkhani Ghiyasvand Student ID : 5990351**

I have worked in Data Processing in Iran as a programmer/analyst for 4 years. My involvement on different projects has varied and has been from hands on development to develop and membership of multiple projects. My activity was on database design, analysis, development and performance management using Microsoft Project and UML and I was maintaining some project on VB and C#.I passed some courses on Reverse Engineering. Based on my experience I will work on UML part and reverse engineering.

**Bahareh Mohammadpourbarghi Student ID : 5894336**

I have worked as frontend developer in a company for 2 years. The languages which I used more in that period were HTML and JavaScript. I have done some projects in Java SE and Java Web as part of my courses in Concordia University. I know the concept of OOP thoroughly so I would be able to comprehend architecture of this project which is in C#. I like the whole idea behind this project which is assignment and event management for students so I will work on UML diagrams related to this project.

**Shahla Noori Student ID : 5972671**

Since, I have been developing in c# for almost 8 years in real industrial environments; I would be able to contribute efficiently to understand the code design and code architecture. Recently, I've been part of a 40-member team who were working on a very huge task which was revising a legacy system. To do so, different refactoring tasks were assigned to us that gave me an acceptable idea. I would try to utilize the gained knowledge in order to find the best solution for our project.

**Robert Saliba Student ID : 6412033**

I'm a senior software developer, possessing in-depth knowledge in building products for the web desktop and mobile applications covering almost every aspect of the software industry and software development life cycle. I Worked in many challenging environment where I used my knowledge in front-end (HTML, Css, JQuery , OO JS, AJAX, JSON) and back-end (PHP, JAVA and C#) to develop robust and efficient code for large scale production applications. I believe this knowledge will help me understanding the solution provided as well as constant monitoring, managing the quality assurance and architecture abstraction or redesign.

**Harcharan Singh Pabla Student ID : 6547702**

I was a junior developer in a company for the last one year. I have hands-on experience in C# .Net and SQL 2008. I have knowledge of web application development in C# - which included object oriented programming and also working with HTML and Javascript in the design layer for User Interface tweaking and modifications. I also have experience in Testing of the developed software application – involving Unit Testing and Integration Testing. I have not worked with Unified Modelling Language in the industry but I am really interested in it so I want to focus on the UML generation and also optimize and enhance the application.

# Persona

| |
|---|
| Nan<br>Age: 18<br>Status: Single<br>Occupation: College Student |
| |
| Nan is an 18 year-old student. Nan has different classes each semester in the College. Each class that he attends has its own professor and assigns various types of deliverable such as assignment, project and paper to him. He used to write all the deliverables descriptions and details in his notebook or type it in Word document. Also in order to remind the deadlines he utilizes his smart phone calendar to add events. He finds it complicated and time-consuming to use different tools to save this information. If he wants to send message to his professor or finds out the professor office hours he must search the college website. On the other hand, he checks the college website frequently to be notified about any events or modifications to the class schedule or any cancellation. |

## Stakeholders:

**-Professor** - are usually the persons who add/creates events to the Calendar like exams, lectures, assignments and deliverables deadlines as well as correcting received assignments and posting the class grades to the students through the system. They can also send email notification to communicate with the students for any new updates, regarding the course or any of the deliverables related subject and answer any email coming from a student.

**-Student** - In this system which is a Student Planner Web Application Students are the main users who will interact with It. Students would like to work with this application wherever they have access to the internet. Students can easily enter all the necessary information of their school in the application and have the broad view of all deliverables, events, classes and professors in one place. Student will benefit from the event reminder functionality of calendar in the application.

**-System Administrator** - A System Administrator is the individual who takes care of the installation of the software on the end-user pc, the privileges and permissions that can be granted to the users and has other non-technical knowledge. The system administrator also addresses basic issues with the system and has troubleshooting skills.

## Actors:

### -Google Calendar

Google Calendar is a free time-management web application offered by Google, where A+ Student planner can leverage the Google Calendar API "**Google Sync**" to create new events, view, add, and drag-and-drop events from one date to another without reloading the page as well as delete existing events, and search for events.

Changes to Google Calendar are immediately visible to all users. This allows new features to be added without user action.

### -System User

System user is a human actor who logs in to the system and can interact with different functionalities of the system. Like creating profile and managing that. Creating a schedule and managing it in a way that it covers the user's needs.

### -Administrator

The administrator is a person who sets the privileges and gives the permissions to the users of the system. It's possible that the administrator to be the System user in some cases. Administrator technically is responsible to configure the initial settings in order to give the users capability of using the system under the predefined criteria.

## Use Cases

### Use Case: Grading

**Actor:** Professor

**Description:** This use case allows to a professor to give grades for one or more courses completed by the professor in the previous term.

This use case begins when the professor selects the "Enter Grades ". The system shows a list of courses that the professor completed in the previous term. By selecting a course the system retrieves a list of all students who were registered for related course and grade information for each. The professor enters a grade (A, B, C, D and F) for each student on the list. The system saves the student's grade for the related course.

### Use Case: Sync with Google Calendar

**Actor:** Any user

**Description:** Sync calendar in this context is a mechanism through which you synchronize your local calendar with your web calendar. In general, whenever you have two (or more) calendars on various places (devices) like your laptop, your desktop, your Smartphone, your server, your web account, SharePoint, etc., you can sync all of them with the one you know is the most current/accurate. This means that at the end, you will have the same calendar copy on all of your devices (places).

The user logs on to her system and accesses Google apps. Google apps displays proper icons. The user selects sync Calendar operation. Google sync checks for user account web calendar for consistency and existence. Google sync reads user's local calendar. Google sync check to see which calendar is more up to date. Google sync informs user as of which

calendar it is going to update. The user verifies the action.
Google sync updates the proper calendar

### Use Case: Manage Professor

**Actor:** System User

**Description:** The system user add Professor with details like Name, Email Address, Contact Information and he/she can edit/manage Professor, filter by Semester and edit the personal information. The Professor information can be then used by the student to match to a class name.

### Use Case: Manage Semester

**Actor:** System User

**Description:** The system user can assign a name and number to the semester.the system user can assign a start date and an end date to the semester.

In case a student wants to change the semester details, he/she can edit the semester name, semester start date and end date.

### Use Case: Manage Class

**Actor:** System User

**Description:** The user requests to manage the class. System will display three options to the user. Add, delete or edit class. The user can add his/her class by entering required information such as class name, schedule, credit number, professor name and etc. the system will add this class schedule to the calendar. The user can edit the classes added before like change the professor name or schedule. The system will save the edited information. The user can delete the class. The system will removed all the information related to that class.

### Use Case: Manage Event

**Actor:** Professor, System User

**Description:** The user requests to manage events. The system will display three options to the user. Add, delete or edit event. The user can add different events available in the categorized list such as exam, quiz, meeting, lectures and presentation. The system will save this event and add it to the calendar. The reminder option can be activated by user in order to be informed for deadlines. The user can edit the events entered before. The system will save all modifications. The professor (user) creates event and the system will add it automatically to the calendar and upload it to the application. The user (student) will receive the notification for this event. The user (student) can view the events. The user can delete the events which he/she added before. The system will remove all the information related to that event.

## Use Case: Send notification

**Actor:** Professor

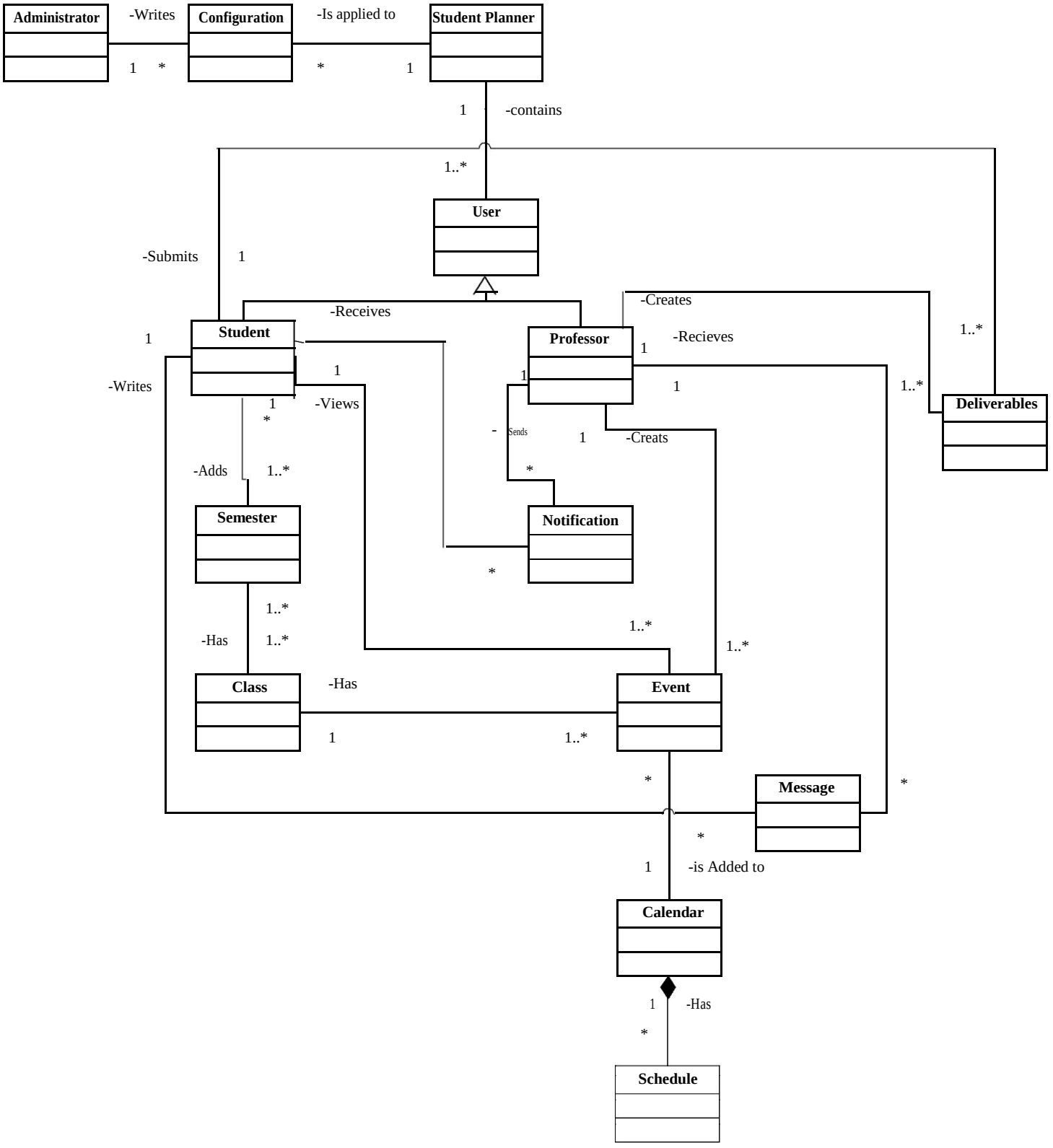
**Description:** The professor selects Create notification. He types the subject of this notification and then the context of that. List of the class's students should already be stored in the system. Professor decides and selects the name of the students to whom this notification should be sent. Professor selects "send" option and the notifications will be forwarded to the selected students. On the student's side there will be an unread notification shown in the notification box.

## Use Case: Managing Deliverables

**Actor:** Student

**Description:** There should be at least one active class that the user is registered in. User selects submit deliverable. List of the user's class is shown. User selects the class he wants to submit the deliverable for. The deadline of the deliverable is shown to the user. User attaches the file and selects the "Upload". By selecting the "send", the attached file will be sent to the professor's Inbox. After selecting the "Send" option user doesn't have any access to delete or modify the attached file.

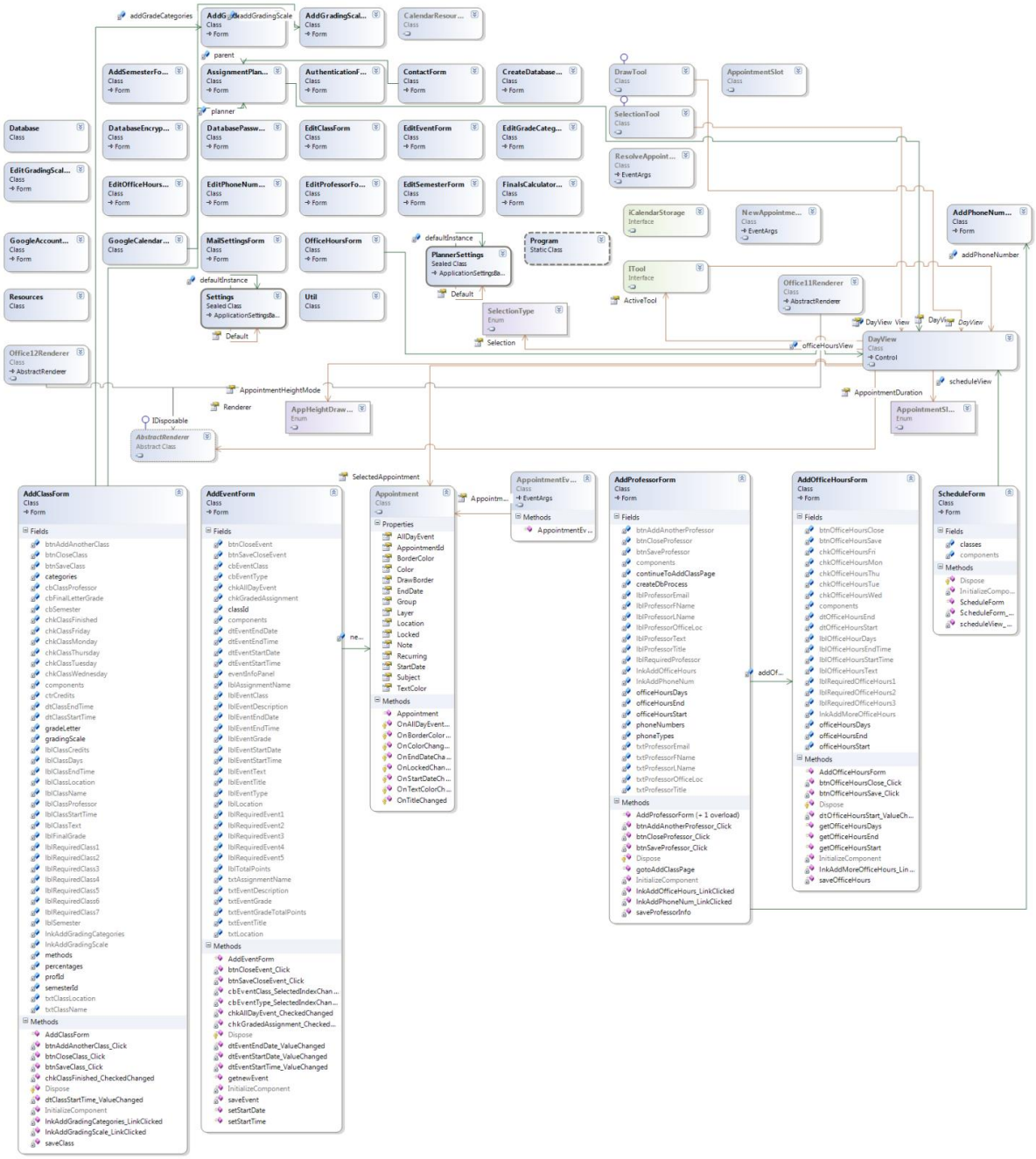
# Ideal Domain Model



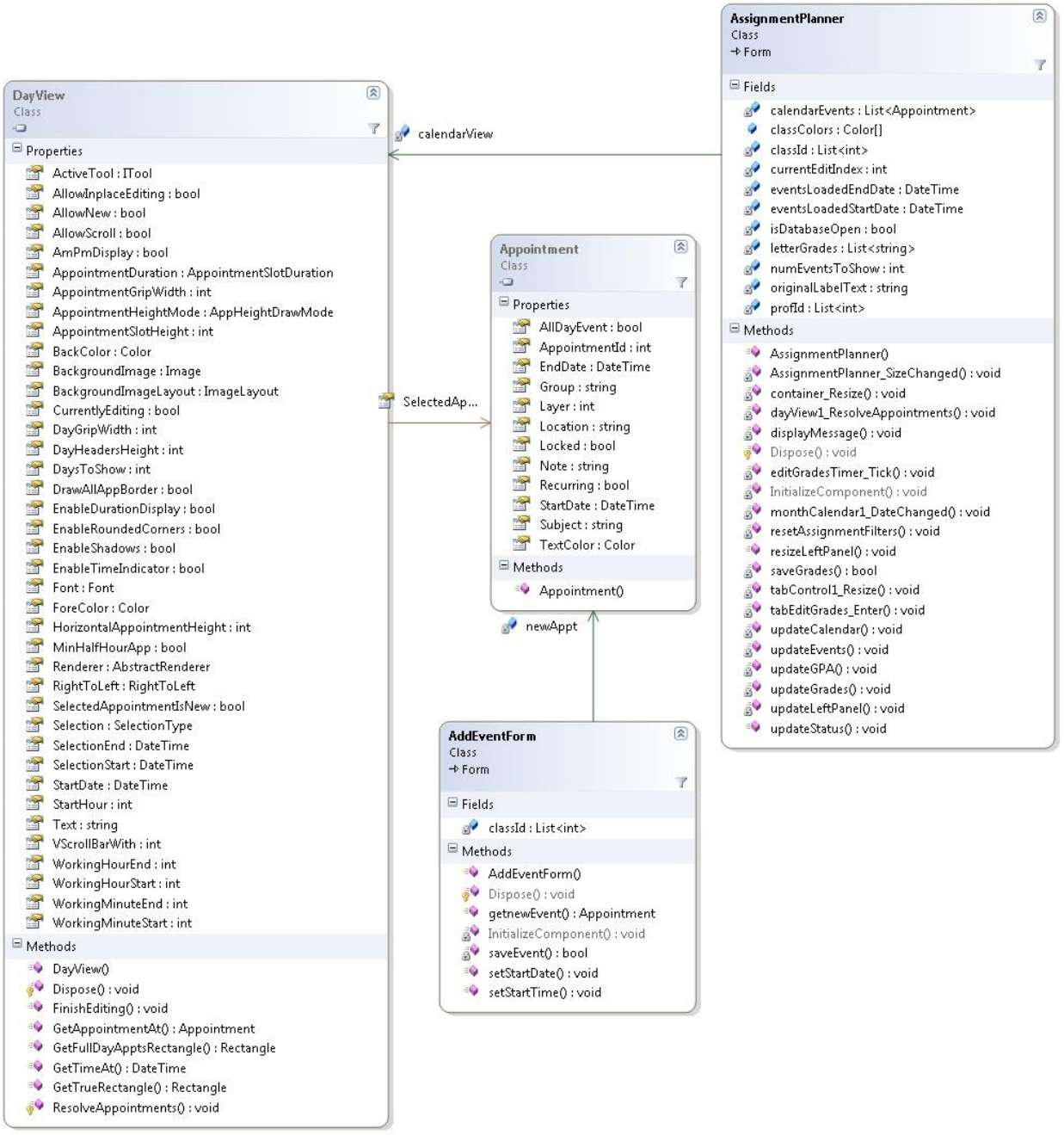
In the domain model which we created for Student Planner, entities are somehow the same as the actual system. In this domain model we have 2 different users who interact differently with the system. Student and Professor are 2 users of our system. Our system has a calendar which contains all schedule information. Whenever professor creates an event it will be stored in calendar in order to be visible to student whenever he/she wants to view it. Professor and students can communicate with each other via messages through the system.

All information about the student semester and classes will be stored in the system. Student can check his/her class schedules in the calendar. Any kind of notifications regarding class cancellation or change of exam date can be created by professor and viewed by students. Any deliverables (projects and assignments) can be assigned for classes by professor and students can view and submit deliverables as well.

# Elaborating on A+ Student Planner Actual Architecture:



The focused part of application architecture is shown below. It has a lot of architectural flaws and code smells.




Our system is a web based application which is totally different from the actual system which is a desktop application. Actual system has only one user (student) while our system can have multiple users at the same time since it is a web-base app. In our system students and professor can communicate via email. Professor can send notifications or messages to multiple students simultaneously. Data in our system is updated via databases automatically but actual system uses local databases and since most users do not want to have to run a local server on their computer in order to run the application, A+ utilizes an embedded version of the database.

Unlike our imagination, the way that A+ Student Planner is designed is less clever than we thought. Since it is not using any specific pattern or even not utilizing the real power of object oriented concepts like polymorphism. Some

significant aspects of a software design like low coupling, high cohesion and information expert is not well applied in the design and it makes it not enough maintainable, less flexible and hard and not clear to understand.
Roughly, all the operations are handled in the UI and it makes the system really low cohesive. For example all the events are handled in the code behind of the forms. While in our design we were going to create a new abstract class that could take care of the event handlers in a way that the UI would fire the events via the specific implementations of that class. In this condition we could keep the UI high cohesive since its responsibility is not controlling the events. And each Handler takes care of very specific actions.
The extracted actual entities are pretty well mapped with the ideal system's entities.
For instance:

| **Actual Classes** | **Conceptual Classes** |
|---|---|
| AddEventForm | Event |
| Appointment | Schedule |
| AddClassForm | Class |
| AddProfessorForm | Professor |
| AddSemesterForm | Semester |
| Assignments | Deliverables |

But there are still some limitations in the actual system; that are managed in the ideal design. As it's mentioned before the actual system intends to limit access to the database to a single user. This is because it intends to develop a typical desktop application that users will be able to run from their computers without being connected to the Internet. Because they wanted to eliminate the dependency on an Internet connection, this restricts the focus to local databases.
Microsoft Visual Studio gives us the ability to extract Class Diagram of the created project, we can add C# classes or namespaces from Architecture Explorer or dependency graphs to a UML class diagram. Also adding C# classes from Solution Explorer is possible. While doing right click on each project that is added to our solution we have an option called "View Class Diagram". Generating class diagram via Visual Studio doesn't support cross project relationship. Each class diagram can show only the classes within the same project. By doing a couple of research in the web we figured out that "ModelingPowerToys" which is a plug-in for visual studio can omit this restriction and lets us create a Class Diagram that shows cross project relationship.

```
public class Appointment : iAppointment
  {
                              #region " Private Fields "
                              private int layer;
                              private string group;
                              private DateTime startDate;
                              private DateTime endDate;
                              ....
                              #endregion

     public Appointment()
     {
                color = Color.White;
                borderColor = Color.Blue;
                Subject = "New Appointment";
     }
     #region " Public Properties "
     public int Layer
     {
        get { return layer; }
        set { layer = value; }
     }
     .....

     public bool AllDayEvent
     {
        get
        {
           return allDayEvent;
        }
        set
        {
           allDayEvent = value;
           OnAllDayEventChanged();
        }
     }

#region iAppointment Members
    public DateTime StartDate
     {
        get
         {
                return startDate;
         }
       set
         {
                startDate = value;
                OnStartDateChanged();
         }
     }
   public DateTime EndDate
      {
         get
            {
              return endDate;
            }
         set
            {
              endDate = value;
              OnEndDateChanged();
            }
      }
                .....
    #endregion
}
```

```
public partial class AddEventForm : Form {
      //store the list of class ids
      private List<int> classId = new List<int>();
       Appointment newAppt = null;
       public AddEventForm()
       {
         InitializeComponent();
         //dynamically add classes to combo box while storing class id's associated with
class name
          Util.addClasses(cbEventClass, classId, true, false, null, null);
          //set start and end date pickers to current date
          dtEventStartDate.Value = DateTime.Now;
          dtEventEndDate.Value = DateTime.Now;
       }
.....
      //get the information from the saved event once the form is closed
      public Appointment getnewEvent() {
        return newAppt;
      }
.....
      private bool saveEvent()
      {
        //ensure user entered a title since it is a required field
        if (txtEventTitle.Text.Equals("") == true) {
          Util.displayRequiredFieldsError("Event Title");
          return false;
        }
        //ensure start time is not later than end time (note this does not apply if an all
day event)
        if (chkAllDayEvent.Checked == false && dtEventStartTime.Value.TimeOfDay >
dtEventEndTime.Value.TimeOfDay) {
          Util.displayError("Invalid Start and End Times", "Error");
          return false;
        }
....
        //begin transaction
        Database.beginTransaction();
.....
        //check if the event is a graded assignment
        if (chkGradedAssignment.Checked == true) {
          //get id of recently inserted event
          object eventID = Database.getInsertedID();
          double grade = Double.MaxValue;
          double gradeTotal = Double.MaxValue;
          //ensure an assignment name was given
          if (txtAssignmentName.Text.Equals("")) {
            Util.displayRequiredFieldsError("Assignment Name");
            return false;
          }
....
        //add event to calendar view
        newAppt = new Appointment();
        newAppt.StartDate = new DateTime(dtEventStartDate.Value.Year,
dtEventStartDate.Value.Month, dtEventStartDate.Value.Day,
dtEventStartTime.Value.Hour, dtEventStartTime.Value.Minute, 0);
        newAppt.EndDate = new DateTime(dtEventEndDate.Value.Year,
dtEventEndDate.Value.Month, dtEventEndDate.Value.Day,
dtEventEndTime.Value.Hour, dtEventEndTime.Value.Minute, 0);
        newAppt.Subject = txtEventTitle.Text;
        newAppt.Note = txtEventDescription.Text;
        newAppt.Location = txtLocation.Text;
        newAppt.AppointmentId = int.Parse(Database.getInsertedID().ToString());
//store unique event id in calendar appointment note
        newAppt.Color = Color.Honeydew;
        newAppt.BorderColor = Color.DarkBlue;
        if (chkAllDayEvent.Checked == true) {
          newAppt.AllDayEvent = true;
          newAppt.EndDate = newAppt.EndDate.AddDays(1);
          newAppt.Color = Color.Coral;
        }
        else if (chkGradedAssignment.Checked == true) {
.....
        return true;
      }
....
}
```

## Code Smells and Possible Refactoring

By looking at the weak design of the Student Planner application, we already knew that there would be a lot of code smells in the code by which applying a proper series of refactorings we could help in improvement of a better design. Regardless of having Dead Code which is scattered in different parts of the code. Here is listed the obvious code smells we noticed:

1. Dispensable code: There is an interface named "iCalendarStorage" and a class named “AppointmentSlot” that are lazy classes since they are not used or implemented in any part of the code. These classes can be removed.

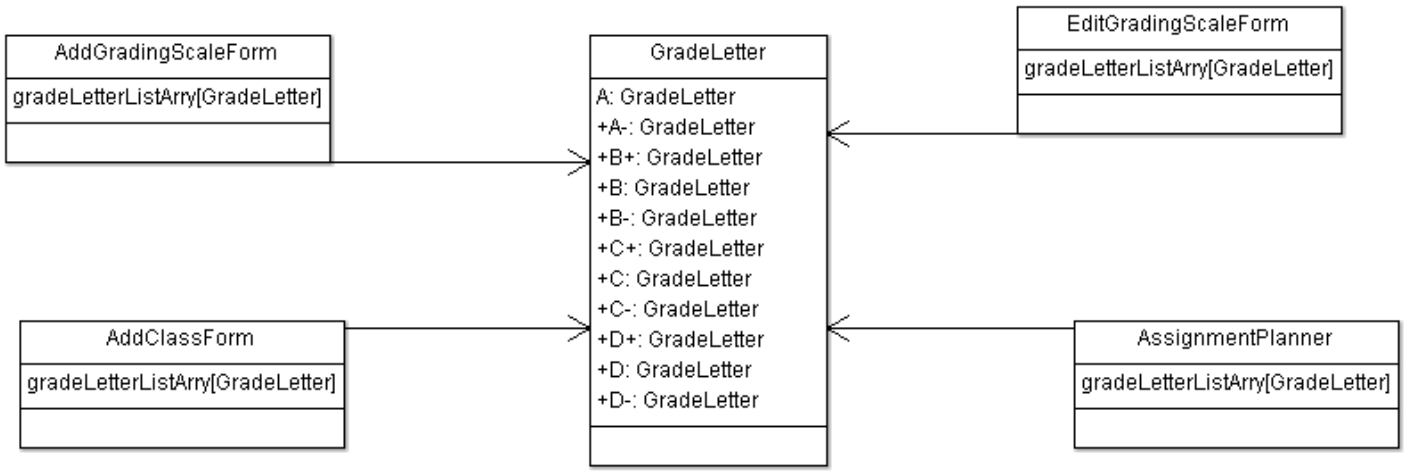
2. Extract Method: In AddEventForm, there is a method “SaveEvent”. It is a very large method, around 78 lines of code. We can use Extract Method and move method refactoring to make the method clear, so it’s easier to read and even modify. It contains nested switch statements and nested if statements inside the switch statements which make the method complex.

3. Low Cohesion: In AddEventForm, the class is performing database Read/Write operations that leads to LOW COHESION. We should have a separate class to tackle database related work.

4. Long Parameter List: In AddClassForm, AddEventForm, EditEventForm and EditClassForm there are couple of methods that have very complex if conditional statements which could be extracted in smaller methods to help reading the methods more easily. So we can use the “Consolidate Conditional Expression” refactoring technique to refactor those if statements.

5. Dead Code/Commented Code - As mentioned above, there is Dead Code at a lot of places in the application. To mention, in file Office12Renderer.cs, there is a block of commented code starting at Line Number 308. If a code block is commented, that means it is no longer used in the execution. So we can simple remove them.

6. AddClassForm.cs has a numeric type code that does not affect its behavior. Here we can replace it with a new class. First we create a new class for the LetterGrade. Then we change the implementation of the source class to use our new class. For all the methods on the source class that uses the code, a new method that uses the new class should be created. In every parts of the code where the old code was used we should modify it in order to use the new class. When making sure that it compiles successfully we should remove the old part that uses the codes. Compile and test is highly recommended to make sure all the changes are done successfully. The figure below is the UML diagram of the expected refactoring.




## Inter-relation between the Code Smells

The AddEventForm has a lot of code smells, which if all rectified one-by-one would lead to a much better code architecture. There is a very long method called SaveEvent. First of all we can extract that method to a new class to make it clearer and increase the modularity. The Save Event is performing some unrelated functions which decreases the cohesion. It is performing database read/write operations which should actually be performed by another entity which takes care of such operations exclusively. So we can extract this functionality from the SaveEvent and create a new class for it. Apart from this, the AddEventForm has a bad smell of long parameter lists, as mentioned above. This can be refactored using the Consolidate Conditional Expression technique and thus make the “if” statements look simpler and more clear. Similarly for EditEventForm, the class is performing unrelated tasks like displaying Error Messages, modifying the Database, etc. Here is shown the EditEventForm class, as an example of the inter-related code smell refactoring’s to make the code better.

```
public partial class EditEventForm : Form {
    private bool saveEvent() {
        //current get event id
        int currentEventId = eventId[cbEvent.SelectedIndex];
        //ensure user entered a title since it is a required field
        if (txtEventTitle.Text.Equals("") == true) {
            Util.displayRequiredFieldsError("Event Title");
            return false;
        }
        //ensure start time is not later than end time (note this does not apply if an all day event)
        if (chkAllDayEvent.Checked == false && dtEventStartTime.Value.TimeOfDay > dtEventEndTime.Value.TimeOfDay) {
            Util.displayError("Invalid Start and End Times", "Error");
            return false;
        }
        //ensure start date is not later than end date (note this does not apply if an all day event)
        if (chkAllDayEvent.Checked == false && dtEventStartDate.Value > dtEventEndDate.Value) {
            Util.displayError("Invalid Start and End Dates", "Error");
            return false;
        }
        //get date in SQLite format
        string startDate = Database.getDate(dtEventStartDate.Value);
        string endDate = Database.getDate(dtEventEndDate.Value);
        .....
        //check if the event is a graded assignment
        if (chkGradedAssignment.Checked == true) {
            //ensure a valid assignment name has been entered
            if (txtAssignmentName.Equals("") == true) {
                Util.displayRequiredFieldsError("Assignment Name");
                Database.abort();
                return false;
            }
        ....
            //if a graded assignment, force user to select class and category
            if (cbEventClass.Text.Equals("") || cbEventType.Text.Equals("")) {
                Util.displayError("Please select a value for both the class and assignment type", "Error");
                Database.abort();
                return false;
            }
            //check that grade and total points are valid number (note that the grade can be empty)
            if ((txtEventGrade.Text.Equals("") == false && double.TryParse(txtEventGrade.Text, out grade) == false) || (txtEventGradeTotalPoints.Text.Equals("")
== false && double.TryParse(txtEventGradeTotalPoints.Text, out gradeTotal) == false)) {
                Util.displayError("Grade and Total Points must be valid decimal numbers", "Invalid Number");
                Database.abort();
                return false;
            }
            if (grade < 0 || gradeTotal < 0) {
                Util.displayError("Grade and Total Points must be positive", "Error");
                Database.abort();
                return false;
            }
        .....
        }
        else {
            //delete graded assignment portion of event
            Database.modifyDatabase("DELETE FROM GradedAssignment WHERE EventID = '" + currentEventId + "';");
        }
        bool containsEvent = eventsHash.ContainsKey(currentEventId);
        Appointment appt;
        if (containsEvent == true) {
            appt = eventsHash[currentEventId];
        }
        else {
            appt = new Appointment();
        }
        //update appointment information
        appt.StartDate = new DateTime(dtEventStartDate.Value.Year, dtEventStartDate.Value.Month, dtEventStartDate.Value.Day,
dtEventStartTime.Value.Hour, dtEventStartTime.Value.Minute, 0);
......
        if (chkAllDayEvent.Checked == true) {
            appt.AllDayEvent = true;
            appt.EndDate = appt.EndDate.AddDays(1);
            appt.Color = Color.Coral;
        }
        else if (chkGradedAssignment.Checked == true) {
        ....
```

**Complex and nested if Conditional Statements**

**Unrelated Operation (Database Query)**