Design Document

AdviseUP Mark II: Web-based scheduling and advising application

Team Members:

Kekai Ariola

Eric Bergquist (Fall Team Lead)

Megan Yamamoto

Industry Representatives:

Brian Toole

Sam Burich

Faculty Advisors:

Dr. Tammy VanDeGrift

Other Contributors:

Dr. Kenneth Lulay (Client)

Ronan Cranley (IS Infrastructure)

Jacob Caniparoli (WAS)
# Table of Contents

Application Design............................................................................................................. 4  
Architecture ....................................................................................................................... 4  
Existing Application Overview ......................................................................................... 5  
Application Database and Stored Procedures ................................................................. 5  
Application ........................................................................................................................ 7  
New Component Structure ............................................................................................... 9  
Materialized Views (P1) ..................................................................................................... 9  
  Stored Procedures ............................................................................................................ 11  
Processor Components .................................................................................................... 14  
  Data Purging (P4) .......................................................................................................... 14  
  Data Encryption of Notes (S4) ...................................................................................... 15  
Interface Components ...................................................................................................... 15  
  Portal Link (P2) ............................................................................................................ 15  
  Checking schedule against completed courses (S1) .................................................... 16  
  Future Enrollment Predictor (S2) .................................................................................. 18  
  New GUI Design (S3) .................................................................................................... 19  
  Reusing Electives For New Templates (S5) .................................................................... 20  
Development Plan ............................................................................................................. 21  
System Test Plan .............................................................................................................. 22  
Updated Milestones .......................................................................................................... 22  
  Final Budget .................................................................................................................. 29  
Conclusion ........................................................................................................................ 30  
References ......................................................................................................................... 31  
Appendix A: Requirements ............................................................................................... 32
Introduction

AdviseUP was created by last year’s Team Foxfire. The goal of AdviseUP is to assist in the advising process for both advisors and advisees. Presently, AdviseUP consists of three main components: a web interface, a processor, and an internal database. The webpages have been developed using ASP.NET and JavaScript, the processor consists of C# scripts, and the application uses a SQL Server database with SQL Server stored procedures.

Team Foxfire created a phenomenal working implementation of AdviseUP whose core design will be retained by Team Hopworks. The major design challenges for Team Hopworks involve the modifications to AdviseUP required to integrate it with the University of Portland live Banner data and Portal website.

The document will focus on the design of the application and the additions and modifications that Team Hopworks will be making. First, an overview of the major components of AdviseUP will be explained including how AdviseUP is currently implemented. Next will be a discussion of how each new feature will be implemented. This includes depictions of how interface pages will be modified and the software components that are involved in the implementation of each feature. After this, the team will explain their design approach and plan for testing of the application. Finally, a list of updated milestones will be included.
Application Design

Architecture

Figure 1 illustrates the existing components - displayed with black boxes and black arrows - and new components - displayed with red boxes and red arrows.

The interface, processor, and application database components of AdviseUP were created by Team Foxfire. The interface allows advisors and advisees to access and plan their schedule over the internet. The application database stores the advisee’s schedules and information about advisors, advisees and courses. The processor links the interface and application database. An overview of the existing components will be covered in the following section. For a more detailed design of these components, please refer to Team Foxfire’s Final Report. Team Hopworks will add functionality to the interface and processor. This is detailed under New Component Structures.

Team Hopworks will also be integrating AdviseUP with the University of Portland Portal website and Banner data. The Portal website provides advisees and advisors with links to the University of Portland’s resources. The team will create a link to AdviseUP in the advising section of the Portal to make it easy to access. The Banner database is the central database of student, faculty, and course information at the University of Portland. Due to concerns about the high volume of traffic that Banner currently handles, AdviseUP will not access the Banner database directly. Instead Brian Toole will be creating materialized views of the Banner data.
required by the application. These views will be stored in an Oracle schema, and will be connected to the internal SQL Server database through a database link.

**Existing Application Overview**

This section of the document will cover the existing application. This section will first go over the application's internal SQL Server database and then the processor and interface components of the application.

**Application Database and Stored Procedures**

The current application database utilizes 13 tables. The table description format is displayed below followed by the description of the 13 tables. These tables are currently populated by Microsoft Excel spreadsheets.

**Format:**

- [Table_Name]: Description (# of fields)
  - Field_Name – Type (max length, if applicable): Description

Example of column name: columnName
Example of table name: [TableName]

-[Advisee]: Stores all information pertaining to advisees (6 fields).

1.) adviseeID – Char(9): The advisee’s 9-digit student number (ex. 001483666)
2.) email – VarChar(15): The advisee’s UP email (ex. ariola13@up.edu)
3.) advisorID – Char(9): The 9-digit UP ID of the advisee’s advisor.
4.) startTerm – Char(6): The term code of when an advisee first entered University of Portland.
5.) firstName – VarChar(30): The first name of the advisee.
6.) lastName – VarChar(30): The last name of the advisee.

-[Corequisite]: Stores information pertaining to co-requisites for courses (2 fields).

1.) coReqID – Char(6): The ID of the courses which is the co-requisite.
2.) courseID – Char(6): The ID of the course which requires the co-requisite.

-[Course]: Stores all information pertaining to courses (5 fields).

1.) courseID – Char(6): The auto-incremented unique ID for the course
2.) dpt – VarChar(4): The department of the course.
3.) num – Char(3): The number of the course.
4.) name – VarChar(30): The name of the course.
5.) credits – real: The number of credits the course is worth.
-[CourseInElective]: Stores information pertaining to courses which may be selected in a drop-down menu (2 fields).

  1.) **courseID** – Char(6): The ID of the course in the elective’s drop down list.
  2.) **electiveID** – Char(6): The ID of the elective.

-[CourseInTemplate]: Stores information about courses that exist in a template (3 fields).

  1.) **templateID** – Char(3): The ID of the template
  2.) **courseID** – Char(6): The ID of the course in the template.
  3.) **term** – Char(6): The location of the course in the template.

-[Elective]: Stores information about electives (2 fields)

  1.) **ElectiveID** – Char(6): The auto-incremented unique ID for the elective.
  2.) **name** – VarChar(20): The name of the elective.

-[ElectiveInSchedule]: Stores information about electives in advisees’ schedules (4 fields).

  1.) **adviseeID** – Char(9): The ID of the advisee.
  2.) **electiveID** – Char(6): The ID of the elective in schedule.
  3.) **selectedCourseID** – Char(6): The ID of the course currently selected in the elective.
  4.) **term** – Char(6): The location of the elective in schedule.

-[Employee]: Stores information about advisors (7 fields)

  1.) **employeeID** – Char(9): The 9-digit UP ID of the employee.
  2.) **email** – VarChar(15): The employee’s UP email
  3.) **isAdministrator** – Boolean: Indicates whether employee is administrator in the application.
  4.) **firstName** – VarChar(30): The first name of the employee.
  5.) **lastName** – VarChar(30): The last name of the employee.
  6.) **location** – VarChar(30): Where the employee can be found on campus.
  7.) **phone** – VarChar(12): The phone number of the employee.

-[Log]: Stores information pertaining to log entries (4 fields).

  1.) **logEntryID** – VarChar(3): The unique auto-incremented ID of the log entry.
  2.) **adviseeID** – Char(9): The ID of the advisee associated with the log entry.
  3.) **author** – VarChar(30): The person (name) who generated the log entry.
  4.) **date** – TimeStamp: The date the log entry was generated.
-[Note]: Stores note information about an advisee (5 fields).

1.) noteID – VarChar(3): The unique auto-incremented ID of the note.
2.) adviseeID – Char(9): The ID of the advisee associated with the note.
3.) note – Text: The text of the note.
4.) author – VarChar(30): The person (name) who generated the note.
5.) date – TimeStamp: The date the note was generated.

-[Prerequisite]: Stores prerequisite information about courses (2 fields).

1.) preReqID – Char(6): The ID of the courses which is the prerequisite.
2.) courseID – Char(6): The ID of the course which requires the prerequisite.

-[Tag]: Stores tag information about courses (2 fields).

1.) courseID – Char(6): The ID of the course.
2.) tag – Integer: The tag associated with the course (1 = Fall, 2 = Spring, 3 = Summer).

-[Template]: Stores information about templates (2 fields).

1.) templateID – Char(3): The unique auto-incremented ID of the template.
2.) name – Char(20): The name of the template.

AdviseUP currently has 68 stored procedures. Stored procedures are objects within the database that run SQL queries when executed. Please refer to Team Foxfire’s Design document for details on these stored procedures.

**Application**

The AdviseUP application uses the languages ASP.NET and C#. The ASP.NET is used to draw and define how the application looks in the browser. The C# creates the shared functions used to organize the data and call the stored procedures to query the student/course information stored in SQL. It also sets the labels in the ASP.NET code. When the application is displaying the graphical view of the course schedule, the C# code creates strings of ASP.NET code that, along with the dragdrop.js file, implements the drag and drop feature.

There are three main sections of the AdviseUP application; the log in screen, advisee page, and advisor page. The log in screen uses the Lightweight Directory Access Protocol to authenticate users with their current university username and password. The advisee page shows an advisee’s schedule as either a list or a box graphical view. The advisor’s page uses the master page and the default page, which contains buttons that appear to the left of the schedule as shown in the advisee page. The advisor’s page to view a specific advisee’s schedule has the same graphical and list view and functions as the advisee’s page. The buttons
that appear on the advisor page open management pages where an advisor can manage and create courses and templates.

![Figure 2: A Screen Shot of AdviseUP](image)

Figure 2 shows a sample student schedule viewable by both advisor and advisee.

Figure 3 shows the page flow for advisees using the program and Figure 4 shows the page flow for advisors.
New Component Structure

These are the new design components that Team Hopworks will implement. Following each header will be the matching functionality code. The functionality table can be seen in Appendix A.

Materialized Views (P1)

Banner is the database that holds student records at the University of Portland. Because Banner holds sensitive material and often experiences heavy user traffic, materialized views of Banner data will be created by Brian Toole that AdviseUP will be able to access. These materialized views will be in an Oracle schema, which is used by the Banner database, and they will be updated nightly. Materialized view descriptions obey the following format:

- [Materialized_View_Name]: Description (#of fields)
  Field_Name – Type (max length, if applicable): Description
Example of column name: columnName
Example of materialized view name: [MaterializedView]

Note: materialized views do not have actual primary or foreign key constraints. These are included as a concept only to show how records will be uniquely identified.

- [Advisee]: This materialized view contains information pertaining to an advisee (7 fields).

1.) adviseIDID – Char(9) (Primary Key): This field contains the advisee ID number of the advisee.
2.) email – VarChar(15): The email of the advisee.
3.) advisorID - Char(9) (Foreign Key): Foreign key into [Employee]; it is the ID of the advisee’s advisor.
4.) startTerm – Char(6): The term code of when an advisee first entered University of Portland.
5.) firstName – VarChar(30): The first name of the advisee.
6.) lastName – VarChar(30): The last name of the advisee.
7.) expectedGraduation – Char(6): The term code of when an advisee is scheduled to graduate.

- [Employee]: This materialized view contains information pertaining to advisors (6 fields). Advisees will not appear in this table.

1.) employeeID – Char(9) (Primary Key): This field contains the ID number of the employee.
2.) email – VarChar(15): The email address of the employee.
3.) firstName – VarChar(30): The first name of the employee.
4.) lastName – VarChar(30): The last name of the employee.
5.) location – VarChar(30): Where the employee can be found on campus.
6.) phone – VarChar(12): The phone number of the employee.

- [Course]: This materialized view contains information pertaining to Banner Course data (4 fields).

1.) dpt – VarChar(4) (Part of Primary Key): The department of the course.
2.) num – Char(3) (Part of Primary Key): The number of the course.
3.) name – VarChar(30): The name of the course.
4.) credits – real: The number of credits the course is worth.

- [Transcript]: This materialized view contains information pertaining to advisees past courses (5 fields).

1.) adviseeID – Char(9) (Part of Primary Key) (Foreign Key): This is a foreign key into [Advisee]; it is the advisee ID number.
2.) dpt – VarChar(4) (Part of Primary Key) (Part of Foreign Key): This is part of the foreign key into [Course]; it is the department of the course.
3.) **num** – *Char(3)* (Part of Primary Key) (Part of Foreign Key): This is the other part of the foreign key into [Course]; it is the course number.

4.) **term** – *Char(6)* (Part of Primary Key): The term code of when the course was taken.

5.) **grade** – *VarChar(2)*: The grade received after completion: pass, low pass, or fail.

- **[Prerequisite]**: This materialized view tracks the prerequisites for a select course (4 fields).

  1.) **dpt** – *VarChar(4)* (Part of Primary Key) (Part of Foreign Key): This is part of the course’s foreign key into [Course]; it is the department of the course.

  2.) **num** – *Char(3)* (Part of Primary Key) (Part of Foreign Key): This is the other part of the course’s foreign key into [Course]; it is the number of the course

  3.) **preDpt** – *VarChar(4)* (Part of Primary Key)( Part of Foreign Key): This is part of the prerequisite course’s foreign key into [Course]; it is the department of the prerequisite course.

  4.) **preNum** – *Char(3)* (Part of Primary Key)( Part of Foreign Key): This is part of the prerequisite course’s foreign key into [Course]; it is the prerequisite course number.

- **[Co-Requisite]**: This materialized view tracks the co-requisites for a select course (4 fields).

  1.) **dpt** – *VarChar(4)* (Part of Primary Key) (Part of Foreign Key): This is part of the course’s foreign key into [Course]; it is the department of the course.

  2.) **num** – *Char(3)* (Part of Primary Key) (Part of Foreign Key): This is part of the course’s foreign key into [Course]; it is the number of the course

  3.) **coDpt** – *VarChar(4)* (Part of Primary Key)( Part of Foreign Key): This is part of the co-requisite course’s foreign key into [Course]; it is the department of the co-requisite course.

  4.) **coNum** – *Char(3)* (Part of Primary Key)( Part of Foreign Key): This is part of the co-requisite course’s foreign key into [Course]; it is the co-requisite course number.

**Stored Procedures**

Each new stored procedure used by AdviseUP is listed below, followed by a description of the procedure. All of the “fetch” procedures will retrieve information from the views of Banner data. When Banner data needs to be added to the internal SQL Server tables the appropriate fetch procedure will be used to retrieve the data, and then it will be inserted into the SQL Server database using pre-existing stored procedures. Procedures are listed according to the following format:

- **Procedure**, which is the name of the procedure;

- **Description**, which describes what the stored procedure is designed to perform;

- **Inputs**, which are parameters passed into the procedure that are necessary for the stored procedure to achieve the desired results;
- **Outputs**, which are data items that are derived from processing a query and returned to the calling program;
- **Query Results**, which is the data retrieved from the database which may also be returned to the calling program;
- **Result Format**, which defines the format of the data to be returned to the calling program;
- **Tables/Views Accessed**, which lists the table and materialized views used in the stored procedure.
- **Fields Accessed**, which lists the fields accessed by the stored procedure.
- **Tables Modified**, lists the tables that are modified by this stored procedure.

1. **Procedure**: fetchAdvisee  
   **Description**: Gets an Advisee’s information from the view of Banner data  
   **Inputs**: adviseeID  
   **Query Results**: The advisee information for the specific advisee  
   **Results Format**: adviseeID, email, advisorID, startTerm, firstName, lastName  
   **Tables/Views Accessed**: [Advisee(view)]  
   **Fields Accessed**: [Advisee(view)]: adviseeID, email, advisorID, startTerm, firstName, lastName  
   **Tables Modified**: N/A

2. **Procedure**: fetchAdvisor  
   **Description**: Gets the information for an advisor  
   **Inputs**: employeeID of the advisee  
   **Query Result**: The advisor’s information from Banner data  
   **Result Format**: employeeID, email, firstname, lastName, location, phone  
   **Tables/Views Accessed**: [Employee(view)]  
   **Fields Accessed**: [Employee(view)]: employeeID, email, firstname, lastName, location, phone  
   **Tables Modified**: N/A

3. **Procedure**: fetchCourse  
   **Description**: Gets the information for a course from the view of Banner data  
   **Inputs**: dpt, num  
   **Query Result**: A record of the course info for that course  
   **Result Format**: dpt, num, name, credits  
   **Tables/Views Accessed**: [Course(view)]  
   **Fields Accessed**: [Course(view)]: dpt, num, name, credits  
   **Tables Modified**: N/A

4. **Procedure**: fetchCoursePrereqs
Description: Gets the list of prerequisite courses for a selected course from the view of Banner data
Inputs: dpt, num
Query Result: A result set giving the list of pre-requisite courses
Result Format: dpt, num
Tables/Views Accessed:
[Prerequisite(view)]
Fields Accessed:
[Prerequisite(view)]: dpt, num, preDep, preNum
Tables Modified: N/A

5. Procedure: fetchCourseCoreqs
Description: Gets the list of co-requisite courses for a selected course from the view of Banner data
Inputs: dpt, num
Query Result: A result set giving the list of co-requisite courses
Result Format: dpt, num
Tables/Views Accessed:
[Co-requisite(view)]
Fields Accessed:
[Co-requisite(view)]: dpt, num, coDep, coNum
Tables Modified: N/A

6. Procedure: getCoursePrediction
Description: Gets the number of advisees with a specific course in their schedule for a given semester.
Inputs: courseID and term
Outputs: Course prediction
Query Results: The number of times a course appears in advisee schedules for a given term.
Results Format: count
Tables/Views Accessed:
[CourseInSchedule]
Fields Accessed:
[CourseInSchedule]: courseID, term
Tables Modified: N/A

7. Procedure: getElectivePrediction
Description: Gets the number of advisees with a specific course in their schedule as an elective for a given semester.
Inputs: courseID and term
Outputs: Course prediction
Query Results: The number of times a course appears in advisee schedules for a given term.
Results Format: count
Tables/Views Accessed:  
[CourseInSchedule]

Fields Accessed:  
[CourseInSchedule]: courseID  

Tables Modified: N/A

8. Procedure: getListOfElective  
Description: Gets all electives created for templates and schedules  
Inputs: None.  
Outputs: List of electiveID’s and names  
Query Results: List of electives are displayed in a scroll box  
Results Format: electives  

Tables/Views Accessed:  
[Elective]  

Fields Accessed:  
[Elective]: electiveID  
[Elective]: name  

Tables Modified: N/A

9. Procedure: verifyCourse  
Description: Verifies that a student took and passed the given course in the given term  
Inputs: adviseeID, depName, courseNum, term  
Outputs: Grade received  
Query Results: The grade that was earned, or an error if the course was not taken  
Results Format: grade as a P for pass, L for low pass, or F for failure.  

Tables/Views Accessed:  
[Transcript(view)]

Fields Accessed:  
[Transcript(view)]: grade  

Tables Modified: N/A

**Processor Components**

**Data Purging (P4)**

The AdviseUP internal database contains every student’s projected schedule. Since this application is intended for long-term use, the team will create procedures to periodically purge the schedules of graduated and inactive students. Without these procedures the application would not be sustainable, since the application database size would continue to grow.

The policy for the purging of data within AdviseUP is as follows. The view of Banner data provided to AdviseUP will contain information about active, currently enrolled students at UP. Therefore if a student is in the internal database but not on the view,
they are not currently an active student at UP. There are many reasons for a student to be no longer active: they may have graduated, withdrawn, or may be spending a semester abroad or taking a semester off. Thus, AdviseUP will not immediately purge inactive students from the internal database. Instead, when a student has their schedule initialized in AdviseUP, the projected graduation year of that student will be stored. At the end of every semester, AdviseUP will check every student in its internal database against the view of Banner data. For every student in the internal database not in the view, i.e. no longer active, AdviseUP will check if it is 2 or more years after the students projected graduation year, and if so, that student’s schedule will be purged from the internal database.

**Data Encryption of Notes (S4)**

AdviseUP allows students and their advisor to store and share notes. With the intended public release of AdviseUP, the contents of these notes may contain sensitive information. To protect this information in case someone with malicious intent gained access to the database, AdviseUP will encrypt the notes that are created and stored in the internal database. The team will be using built-in SQL data encryption to encrypt the note field of the notes table in the application database.

**Interface Components**

**Portal Link (P2)**

Because Portal is used by almost every member of the University, a link will increase the availability and usage of AdviseUP. Figure 2 shows a portal link for advisees.
Team Hopworks will be working with the Portal administrator, Jacob Caniparoli, to put what will most likely be a HTML link to the application website within the ‘Advising’ portlet. The team will also be working with Ronan Cranely to change the AdviseUP authentication from Lightweight Directory Access Protocol (LDAP) to Central Authentication Service (CAS). This switch will allow users to access AdviseUP without needing to enter their username and password again. Figure 3 shows what the links will look like for advisors.

![Figure 3: AdviseUP authentication change](image)

Figure 6: AdviseUP links from Advisor Portal View

Instead of the one link to AdviseUP that students have, advisors will have a link next to each students email address. Each link takes advisors directly to the AdviseUP schedule of the associated advisee.

**Checking schedule against completed courses (S1)**

The current AdviseUP application checks if prerequisites or co-requisites have not been fulfilled based on how a student has planned their four-year schedule. With the inclusion of Banner data, Team Hopworks will also compare student’s classes from previous semesters with their transcript from Banner data so that courses in their schedule that they did not take or pass are flagged with a warning. See Figure 4 below for what warning flags will look like.
New Features:

- Warning flags colors and associated meaning:
  - Red = failed (new)
  - Yellow = low pass (new)
  - Gray = not taken (new)
  - Magenta = prerequisite / co-requisite violation (recolored from red)

 Stored Procedures

1. verifyCourse(adviseeID, depName, courseNumber, term)

When a course is moved in a student’s schedule, the stored procedure verifyCourse will be called. Since we can only verify courses taken and completed, if the term is the current term or a future term the procedure will return nothing. If the term is a past term, the stored procedure will look in the materialized Transcript view of Banner data for the record of that student taking that course in that term. If there is no match, the procedure will return an error. If such a record exists, one of three results may be returned: high pass, low pass or fail. If anything other than a high pass is returned, the application will proceed to mark the course with a warning flag. It is the student’s responsibility to rearrange their schedule to remove the warnings. Once the schedule has been modified, the warning flags will be removed as shown in Figure 4.
Future Enrollment Predictor (S2)

A feature requested by Dr. Tammy VanDeGrift is a Future Enrollment Prediction. Currently, administrators have no way of predicting how many students are interested in a specific class for any given semester. On the ‘Manage Course’ page from the advising side of the application, advisors can select a year and term and view the number of students planning to take that course at that time. See Figure 3 below.

Figure 8: Warning flags removed

For more details on this screen’s other features, refer to the Team Foxfire Final Report.
New Features:
- ‘Predict Enrollment’ button

Stored Procedures
1. `getCoursePrediction(courseID, term)`
2. `getElectivePrediction(courseID, term)`

Advisors select a year and term from the respective drop down menus once they have selected a course to manage. The ‘Predict Enrollment’ button is created within the ManageCourse.aspx file. When the button is clicked, the method `Prediction_Click` from ManageCourse.aspx.cs is invoked. The shared function `convertToTermCode` from SharedFunctions.cs combines the desired year and semester into the term code. The courseID and term code are used as the arguments for stored procedures 1 and 2. Both procedures are necessary since the application database stores courses chosen to fill an elective in a students schedule separately from courses in their schedule. The results from the stored procedures are summed and displayed to the right of the ‘Predict Enrollment’ button.

**New GUI Design (S3)**

Since AdviseUP will be open to the campus upon completion, the color scheme and design of the GUI for the application will be changed to be more in line with other University of Portland websites and applications. The color scheme will be modified to follow those outlined in the *University of Portland Brand Book.*
Reusing Electives For New Templates (S5)

Every time an advisor adds an elective to a template, they currently must create the elective, even if this elective is present in another template. We will allow advisors to search preexisting electives and add them to templates. Figure 10 shows how an advisor might create an elective.

![Figure 10: Creating an Elective](image)

This page is pre-existing under the ‘Add Elective’ page. We will change this page to be the ‘Create Elective’ page. The ‘Add Elective’ page will now look similar to the ‘Add Course’ page. Figure 11 shows what the new ‘Add Elective’ page will look like.
New Features:

- Add Elective Page

Stored Procedures

1. `getListOfElectives()` (new)
2. `getElectiveChoices(electiveID)` (existing)
3. `insertElectiveIntoTemplate(electiveID,templateID,selectedCourseID,term)` (existing)

The `getListOfElectives` procedure is used to populate the ‘Elective’ scroll box. When an advisor selects an elective, the ‘Elective Options’ scroll box is populated by the `getElectiveChoices` procedure. When an advisor wishes to add the elective to the template, the advisor clicks ‘add course’ and `insertElectiveIntoTemplate` is called.

### Development Plan

For the development of AdviseUP Mark II, the team will be utilizing the agile software development approach. The agile software development approach focuses on teamwork, collaboration, and the frequent delivery of a working application to the client. This approach is the most beneficial to Team Hopworks since this project involves improving an already existing application rather than starting a new project from the ground up. The changes to AdviseUP detailed in this document will be completed in sections. The team will focus on completing a specific requirement and producing a working application with that functionality before moving to the next improvement. The most crucial and most difficult functionality will
be integrating AdviseUP Mark II with the live Banner database. For this, the team will be working with Brian Toole to create the views of Banner data that will be used in this application. For these reasons, the team will be implementing this section first.

For this project, the team will be utilizing pair programming. This will ensure that every member understands the changes being made and the likelihood of creating an error is minimized.

Responsibility for the components will be split between members of the team. Megan Yamamoto will be primarily responsible for the changes and additions to the user interface, as well as the creation of Portal links to AdviseUP. Kekai Ariola and Eric Bergquist will be responsible for the integration of the database link to Banner data and the modifications to the internal database template.

**System Test Plan**

As described in the development plan above, each feature that the team implements will be completed and tested according to its specifications before moving on to the next functionality. To this effect the team has a milestone for the testing of each feature soon after its completion. With this the team will have a working application after each new improvement is implemented.

Before its final release, the team will beta test the application by asking the faculty and students to use the application. The team will be requesting the faculty’s input via e-mail that will contain a link and a response sheet. For the students, the team will be hosting a beta testing session in Shiley where the team will provide pizza to students who participate in the beta testing session.

**Updated Milestones**

The list of project milestones has been modified to reflect Team Hopworks plan to link to Banner data through a database link and the Team’s greater understanding of the order in which functionalities should be completed. All milestones completed as of the most recent version of the design document list their actual completion date and are marked with an asterisk.

Table 1 lists the class-oriented milestones of Team Hopworks. These are the presentations and documents that the team must create. Table 2 lists the project milestones, which is a list of the tasks the team needs to complete for the project. Each project milestone in the table has a description of the milestone, the deadline, and what function of AdviseUP the milestone addresses. A description of each project milestone is listed below the table.
### Table 1: Class Milestones

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Completion Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Functional Design Document Draft</td>
<td>24 Sep 2012*</td>
</tr>
<tr>
<td>2</td>
<td>Functional Design Document Final Version</td>
<td>26 Oct 2012*</td>
</tr>
<tr>
<td>3</td>
<td>September Program Review (Megan)</td>
<td>28 Sep 2012*</td>
</tr>
<tr>
<td>4</td>
<td>Design Document Draft</td>
<td>5 Nov 2012*</td>
</tr>
<tr>
<td>5</td>
<td>Design Document Final Version</td>
<td>16 Nov 2012</td>
</tr>
<tr>
<td>6</td>
<td>October Program Review (Kekai)</td>
<td>28 Oct 2012*</td>
</tr>
<tr>
<td>7</td>
<td>November Program Review (Eric)</td>
<td>30 Nov 2012</td>
</tr>
<tr>
<td>8</td>
<td>January Program Review (Megan)</td>
<td>25 Jan 2013</td>
</tr>
<tr>
<td>9</td>
<td>February Program Review (Kekai)</td>
<td>22 Feb 2013</td>
</tr>
<tr>
<td>10</td>
<td>Final Program Review</td>
<td>7 Mar 2013</td>
</tr>
<tr>
<td>11</td>
<td>Final Report First Draft</td>
<td>15 Mar 2013</td>
</tr>
<tr>
<td>12</td>
<td>Final Report Final Version</td>
<td>2 Apr 2013</td>
</tr>
<tr>
<td>13</td>
<td>Founder's Day Presentation</td>
<td>9 Apr 2013</td>
</tr>
<tr>
<td>14</td>
<td>Post Mortem Presentation</td>
<td>18 Apr 2013</td>
</tr>
</tbody>
</table>
**Table 2: Project Milestones**

<table>
<thead>
<tr>
<th>Milestone</th>
<th>Description</th>
<th>Completion Date</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Hold AdviseUP code review with Sam Burich</td>
<td>5 Oct 2012*</td>
<td>All</td>
</tr>
<tr>
<td>B</td>
<td>Identify what data must be drawn from Banner</td>
<td>8 Oct 2012*</td>
<td>P1</td>
</tr>
<tr>
<td>C</td>
<td>Registrar approval to use Banner database</td>
<td>12 Oct 2012*</td>
<td>All</td>
</tr>
<tr>
<td>D</td>
<td>Complete policy for purging data</td>
<td>24 Oct 2012*</td>
<td>P4</td>
</tr>
<tr>
<td>E</td>
<td>Complete code for future course enrollment prediction feature</td>
<td>31 Oct 2012*</td>
<td>S2</td>
</tr>
<tr>
<td>F</td>
<td>Complete requirements for views of Banner database</td>
<td>7 Nov 2012*</td>
<td>P1</td>
</tr>
<tr>
<td>G</td>
<td>Complete CAS authentication code</td>
<td>12 Nov 2012</td>
<td>P2</td>
</tr>
<tr>
<td>H</td>
<td>Complete code to modify templates</td>
<td>23 Nov 2012</td>
<td>P3</td>
</tr>
<tr>
<td>I</td>
<td>Test CAS authentication</td>
<td>23 Nov 2012</td>
<td>P2</td>
</tr>
<tr>
<td>J</td>
<td>Test views of Banner database</td>
<td>28 Nov 2012</td>
<td>P1</td>
</tr>
<tr>
<td>K</td>
<td>Test course prediction and template procedures</td>
<td>28 Nov 2012</td>
<td>S2, P3</td>
</tr>
<tr>
<td>L</td>
<td>Write template guide document for advisors</td>
<td>5 Dec 2012</td>
<td>P3</td>
</tr>
<tr>
<td>M</td>
<td>Client approval for template functionality</td>
<td>25 Jan 2012</td>
<td>P3</td>
</tr>
<tr>
<td>N</td>
<td>Complete code that adds and drops data from internal database as needed</td>
<td>11 Jan 2013</td>
<td>P1</td>
</tr>
<tr>
<td>O</td>
<td>Complete code for portal links to AdviseUP</td>
<td>18 Jan 2013</td>
<td>P2</td>
</tr>
<tr>
<td>P</td>
<td>Test adding and dropping internal data</td>
<td>25 Jan 2013</td>
<td>P1</td>
</tr>
<tr>
<td>Q</td>
<td>Complete code to check student schedules against Banner data</td>
<td>25 Jan 2013</td>
<td>S1</td>
</tr>
<tr>
<td>R</td>
<td>Test the checking of student schedules against Banner data</td>
<td>1 Feb 2013</td>
<td>S1</td>
</tr>
<tr>
<td>S</td>
<td>Complete code for purging data of graduated students</td>
<td>1 Feb 2013</td>
<td>P4</td>
</tr>
<tr>
<td>T</td>
<td>Complete code to modify data to test purging of graduated student data</td>
<td>8 Feb 2013</td>
<td>P4</td>
</tr>
<tr>
<td>U</td>
<td>Modify AdviseUP GUI to conform to UP style guide</td>
<td>8 Feb 2013</td>
<td>S3</td>
</tr>
<tr>
<td>V</td>
<td>Modify AdviseUP to remove outdated functionality</td>
<td>8 Feb 2013</td>
<td>P1</td>
</tr>
<tr>
<td>W</td>
<td>Complete code for encrypting notes</td>
<td>15 Feb 2013</td>
<td>S4</td>
</tr>
<tr>
<td>X</td>
<td>Code Complete</td>
<td>15 Feb 2013</td>
<td>All</td>
</tr>
<tr>
<td>Y</td>
<td>Test procedures for purging graduated students data</td>
<td>20 Feb 2013</td>
<td>P4</td>
</tr>
<tr>
<td>Z</td>
<td>Test procedures for accessing Banner data</td>
<td>20 Feb 2013</td>
<td>P1</td>
</tr>
<tr>
<td>AA</td>
<td>Test portal links to AdviseUP</td>
<td>25 Feb 2013</td>
<td>P2</td>
</tr>
<tr>
<td>BB</td>
<td>Write Users Guide and Maintenance Guide</td>
<td>1 Mar 2013</td>
<td>All</td>
</tr>
<tr>
<td>CC</td>
<td>User and Faculty Testing Complete</td>
<td>8 Mar 2013</td>
<td>All</td>
</tr>
<tr>
<td>DD</td>
<td>Registrar Final Approval for live Banner database</td>
<td>15 Mar 2013</td>
<td>P1</td>
</tr>
<tr>
<td>EE</td>
<td>IS Approval for portal links</td>
<td>22 Mar 2013</td>
<td>P2</td>
</tr>
<tr>
<td>FF</td>
<td>Code frozen after integration testing</td>
<td>5 Apr 2013</td>
<td>All</td>
</tr>
</tbody>
</table>

**A. Hold AdviseUP code review with Sam Burich**

The team will meet with Sam Burich to review the code written for AdviseUP last year.
B. Identify what data must be drawn from Banner

The team will determine what information we will need from the live database and give this to Brian Toole so he can write the queries we will need.

C. Registrar approval to use Banner database

The Registrar will be given a demonstration of AdviseUP so they will allow Team Hopworks to use the Banner database.

D. Complete policy for purging data

The team will decide under what conditions an academic plan will be purged from the database.

E. Complete code for future course enrollment prediction feature

Code for the future course enrollment prediction feature will be complete.

F. Complete requirements for views of Banner database

The specifications of the Banner data the application needs will be completed and given to Brian Toole of WAS so that he can create views of this data that the application can access.

G. Complete CAS authentication code

AdviseUP will be updated to use CAS authentication.

H. Complete code to modify templates

The code to implement the modification of existing templates will be completed.

I. Test CAS authentication

Test the CAS authentication of AdviseUP. Code will be updated if necessary.

J. Test views of Banner database

Test the database link and ensure that the application can access all the information required. Code will be updated if necessary.
K. Test course prediction and template procedures

The team will test the course prediction feature by generating fake student schedules. The template features will also be tested, and code will be updated if necessary.

L. Write template guide document for advisors

A guide for proper template creation, editing, and modification will be written for advisors use.

M. Client Approval for template functionality

The team will meet with our client, Professor Lulay, and demonstrate the functionality he requested.

N. Complete code to add and drop data from internal database

The team will complete the code so that when a user begins a session the views of Banner data are queried for the student's information, which is then added to the internal database. When the session is completed the information will be dropped.

O. Complete code for portal links to AdviseUP

The code for generating and adding a link to AdviseUP in the advising portlet will be completed and tested using the portal test server.

P. Test adding and dropping internal data

The team will test that information is being correctly added to the internal database when user sessions begin and dropped once the session ends and the data is no longer needed. Code will be updated if necessary.

Q. Complete code to check student schedules against Banner data

Code for checking student schedules against the Banner database to verify students took and passed their intended classes for previous semesters.

R. Test the checking of student schedules against Banner data

Test that student schedules are being correctly checked against the Banner data. Code will be updated if necessary.
S. Complete code for purging data of graduated students

The code to purge graduated students will be complete.

T. Complete code to modify data to test purging graduated student data

Complete code that will modify data pulled from Banner in order to test the purging of graduated student data.

U. Modify AdviseUP GUI to conform to UP style guide

The AdviseUP GUI will be modified so that it looks similar to the portal and other UP web pages.

V. Modify AdviseUP to remove outdated functionality

With the database link to Banner's live data, some of the functionality implemented by Team Foxfire is no longer required and will be removed.

W. Complete code for encrypting notes

The internal database will be modified so that the notes are stored in an encrypted format.

X. Code Complete

All coding for the additional functionality of AdviseUP Mark II will be complete.

Y. Test procedures for purging graduated student data

Test that AdviseUP is correctly purging data. Code will be updated if necessary.

Z. Test procedures for accessing Banner data

The team will test to ensure that all queries to the Banner database are working as intended after integration with the live database. Code will be updated if necessary.

AA. Test portal links to AdviseUP

The links to AdviseUP generated in the portal will be tested by the team and by the portal administrators. Code will be updated if necessary.
BB. Write Users Guide and Maintenance Guide

The team will write a guide for users of AdviseUP. A maintenance guide will also be written for Web and Administrative Systems so that they can maintain AdviseUP.

CC. User and Faculty Testing Complete

Testing sessions with faculty and students will be completed.

DD. Registrar Final Approval for live Banner data

The Registrar’s office will have given their approval for the completed AdviseUP application to access the live Banner database and be used by students.

EE. IS Approval for portal links

Information Services will have given their approval for the links to AdviseUP to appear on the portal pages of advisors and advisees.

FF. Code is frozen after Integration Testing.

After all components have been tested after the integration with the live Banner data and Portal links, the AdviseUP code will be frozen.
Final Budget

Table 3 indicates the budget for developing AdviseUP Mark II. Because the development software is free and the servers are provided by Information Services, the main expenses are food and drinks to provide to students testing the system.

Table 3: Project Budget

<table>
<thead>
<tr>
<th>Line</th>
<th>Category</th>
<th>Description</th>
<th>Number</th>
<th>Rate</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Printing</td>
<td>A hard copy of each of the previous team’s documents</td>
<td>245</td>
<td>$0.05</td>
<td>$12.25</td>
</tr>
<tr>
<td>2</td>
<td>Food</td>
<td>Large 1 topping pizza</td>
<td>10</td>
<td>$7</td>
<td>$70</td>
</tr>
<tr>
<td>3</td>
<td>Accessories</td>
<td>Plates and napkins</td>
<td>3</td>
<td>$5</td>
<td>$15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TOTAL</td>
<td></td>
<td></td>
<td>$117.25</td>
</tr>
</tbody>
</table>
Conclusion

AdviseUP Mark II will improve AdviseUP by connecting it to live Banner data and incorporating faculty requested functionality. The internal application database will no longer be populated by Excel spreadsheets, but instead access a view of live data through a database link set up by Brian Toole. A Portal link to AdviseUP will increase the application’s visibility so it will be more convenient to access and, hopefully, used by more students and advisors. Other functionality involves the addition of C# methods and stored procedure calls within the application.

The team plans to test the application iteratively as each function is implemented. When all major components and functionalities are complete, the application will be tested by students and faculty.

Team Hopworks is confident that their work on AdviseUP will result in an application ready to be used by all schools within the University of Portland. The team will work with the Registrar and WAS to ensure that AdviseUP is approved and will be hosted and maintained so it can be used by the University of Portland community.
References


# Appendix A: Requirements

## Primary Functionality

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>AdviseUP uses SQL queries to pull current data from the University of Portland’s live Banner database and store in its application database.</td>
</tr>
<tr>
<td>P2</td>
<td>The AdviseUP link is accessible by all students and faculty across campus through the advising portlet on the portal.</td>
</tr>
<tr>
<td>P3</td>
<td>Advisors can copy a template and then modify it instead of having to create a new template for every discipline variation.</td>
</tr>
<tr>
<td>P4</td>
<td>AdviseUP will purge sensitive material after every user session. Academic schedules will be purged after a student has graduated. Graduation data will be received from Banner.</td>
</tr>
</tbody>
</table>

## Secondary Functionality

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>AdviseUP crosschecks advisees’ completed courses with the previous semesters on their academic schedules. Advisors and advisees receive visual and textual warnings if the Advisee has not completed a course on their academic schedule or if they have not completed the required prerequisites and co-requisites to take a given course.</td>
</tr>
<tr>
<td>S2</td>
<td>AdviseUP calculates how many students plan to take any given class for any given semester. Advisors can view a future enrollment prediction.</td>
</tr>
<tr>
<td>S3</td>
<td>The Graphical User Interface of AdviseUP matches the look and feel of other UP web pages and applications.</td>
</tr>
<tr>
<td>S4</td>
<td>Notes on AdviseUP will be encrypted when being stored in the application database.</td>
</tr>
<tr>
<td>S5</td>
<td>Existing electives can be searched and added when editing or creating a template.</td>
</tr>
</tbody>
</table>