Crosshair Solutions

Delivering Software with Pinpoint Accuracy

Software Requirements Specification

For

Whiteboard™

Version 2.20 approved

Prepared By: Jason Boyle
Mike Murray
Nick Patel
Nestor Reyes
Wilson Lau
Dana Goyette

Crosshair Solutions

April 5, 2009

Contact Information: Mike Murray (Customer Liaison) - **mimurray@calpoly.edu**
http://wiki.csc.calpoly.edu/crosshairsolution**s**

Credits

Section 1: Nestor Reyes
Section 2: Jason Boyle
Section 3: Wilson Lau
Section 4: Wilson Lau
Section 5: Mike Murray
Section 6: Nick Patel
Appendix A: Nick Patel
Appendix B: Dana Goyette
Appendix C: Jason Boyle
Appendix D: Nick Patel
Appendix E: Dana Goyette
Appendix F: Nick Patel
Appendix G: Mike Murray
Appendix H: Jason Boyle
Appendix I: Dana Goyette
Appendix J: Jason Boyle
Appendix K: Nestor Reyes

Table of Contents

Credits ii
Table of Contents iii
1. Introduction 1
1.1 Purpose 1
1.2 Document Conventions 1
1.3 Intended Audience and Reading Suggestions 1
1.4 Project Scope 1
1.5 References 2
2. Overall Description 3
2.1 Product Perspective 3
2.2 Top Level Data Flow Diagram 3
2.3 Operating Environment 4
2.4 Design and Implementation Constraints 4
2.5 User Documentation 4
2.6 Assumptions and Dependencies 4
3. System Features 5
3.1 Add, Update, and Delete Courses 5
3.2 View Courses 7
3.3 Add, Update, and Delete Student 8
3.4 View Student 10
3.5 Add, Update, Delete, and View TAs 11
3.6 Add, Update, and Delete Announcements 13
3.7 View Announcements 15
3.8 Add, Update, and Delete Assignments 16
3.9 View Assignments 18
3.10 View and Edit Grades 19
3.11 Set Curve 21
3.12 View Metrics 22
3.13 View Solutions 23
3.14 Upload Solution 24
3.15 View Hand-Ins 25
3.16 Upload Hand-Ins 26
3.17 Grade Estimator 27
4. External Interface Requirements 29
4.1 User Interfaces 29
4.2 Hardware Interfaces 29
4.3 Software Interfaces 29
4.4 Communications Interfaces 29
5. Nonfunctional Requirements 30
5.1 Accuracy 30
5.2 Aesthetics 30
5.3 Compatibility 30
5.4 Availability 30
5.5 Usability 30
5.6 Understandability 30
5.7 Speed 31
5.8 Maintainability 31
5.9 Security 31
5.10 Coding Standards 31
6. Data Dictionary 32

Appendix A: Glossary **34**
Appendix B: Data Flow Models **35**
1. Professor and TA 35
2. Student 36
Appendix C: User Interface Prototypes **37**
Appendix D: Market Research **40**
Appendix E: Comparison Matrix **49**
Appendix F: Vision and Scope **50**
1. Business Requirements **50**
1.1 Background 50
1.2 Business Opportunity 51
1.3 Business Objectives and Success Criteria 51
1.4 Customer or Market Needs 51
1.5 Business Risks 52
2. Vision of the Solution **53**
2.1 Vision Statement 53
2.2 Major Features 53
2.3 Assumptions and Dependencies 53
3. Scope and Limitations **54**
3.1 Scope of Initial and Subsequent Releases 54
3.2 Limitations and Exclusions 54
4. Business Context **55**
4.1 Stakeholder Profiles 55
4.2 Project Priorities 56
4.3 Operating Environment 56
Appendix G: Nonfunctional Requirements Notes **57**
1. Compatible Format 57
2. Justification of Design for a Tabbed Interface 58
Appendix H: Use Cases **59**
1.0 View Courses 58
2.0 View TAs 63
3.0 View Students 67
4.0 View Announcements 73
5.0 View Assignments 77
6.0 View CourseGrades 82
7.0 View Student Grades 84
8.0 Set Curve 85
9.0 View Solutions 86
10.0 Upload Solutions 87
11.0 View Hand-Ins 88
12.0 Upload Hand-Ins 89
13.0 Grade Estimator 90
Appendix I: Class Diagram **91**
Appendix J: Use Case Diagrams **92**
Appendix K: Check List **95**

1. Introduction

1.1 Purpose

This Software Requirements Specification describes the software functional and nonfunctional requirements for release 1.0 of the grading tool Whiteboard™. This document is intended to be used by the project team that will implement the product and verify correct System functionality. All requirements listed here are intended for product release 1.0.

1.2 Document Conventions

All glossary terms will be in *italics*, their definitions can be found in Appendix A: Glossary. All data items will be in **bold**. Definitions for data items can be found in the Data Dictionary (Section 6). For more information, see Appendix J: Class Diagram. This document shall use the third-person male form for pronouns describing users. This document shall use decimal notation for numbering tables.

1.3 Intended Audience and Reading Suggestions

The intended readers of this document are the **student**s of *Cal Poly*, developers enrolled in CPE 309 for the spring 2009 term, Dr. Turner, and our customer, Lauren Tsung. The document is intended to be read in sequential order from Introduction (Section 1) to Data Dictionary (Section 6). Appendices for further reference are provided. The customer will find System Features (Section 3) the most pertinent. Developers will need to look over Sections 2, 3, 4, 5, and 6for the overall description, features, and requirements of the product.

1.4 Project Scope

Refer to Appendix F: Vision and Scope.

1.5 References

1. Boyle, Jason. Murray, Mike. Patel, Nick. *Vision and Scope Document for Whiteboard™, Version* 2.07.

2. ITS Core Accounts. Accounts and Passwords. Cal Poly Service Desk Information Technology Services. 18 January 2008. 7 February 2009. <http://www.servicedesk.calpoly.edu/accounts_passwords/index.html>

3. Cal Poly Password Manager. Computing Support. Cal Poly Service Desk. 22 August 2008. 7 February 2009.<http://servicedesk.calpoly.edu/computing_support/faq/password.html#rules>

4. Simple Mail Transfer Protocol. RFC 5321. October 2008. 7 February 2009. <http://tools.ietf.org/html/rfc5321>

5. Cal Poly Course Catalog. 20 November 2008. 7 February 2009. <http://www.calpoly.edu/~acadprog/>

6. FERPA Use and Release of Student Information Policy. Cal Poly Office of Academic Records. 20 November 2008. 7 February 2009. <http://www.ess.calpoly.edu/_records/stu_info/ferpa_use.htm>

7. W3C Content Web Accessibility Guidelines (WCAG) version 2.0. 11 December 2008. 10 February 2009.<http://www.w3.org/TR/WCAG20/>

8. HTML 4.01 Specifications. 24 December 99. 10 February 2009. <http://www.w3.org/TR/html401/>

9. Cal Poly University Colors. Cal Poly Web Authoring Resource Center. 31 October 2008. 22 February 2009. <http://warc.calpoly.edu/universityid/colors.html>

10. Dalbey, John. “Dr. Dalbey's Java Coding Standard.” 15 June 2008. 23 February 2009. <http://users.csc.calpoly.edu/~jdalbey/SWE/code_std.html>

11. SQLite Home Page. 21 October 2008. 9 April 2009. < http://www.sqlite.org/>

12. Grading Symbols, Academic Records. Cal Poly. 13 November 2008. 2 March 2009. <http://www.ess.calpoly.edu/_records/stu_info/symbols.htm>

13. Grader Baseline Features. CPE308 Customer Page. 23 January 2009. 7 March 2009. <http://turner308customer.pbwiki.com/Grader-Baseline-Features>

2. Overall Description

2.1 Product Perspective

The Whiteboard™ grading tool is a new System that will allow **professor**s and **student**s to organize their **assignment**s, **grade**s, and **announcement**s all in one tool at *Cal Poly*, San Luis Obispo. Figure 1 illustrates both system interfaces and external entities that will be present for release 1.0.

2.2 Top Level Data Flow Diagram

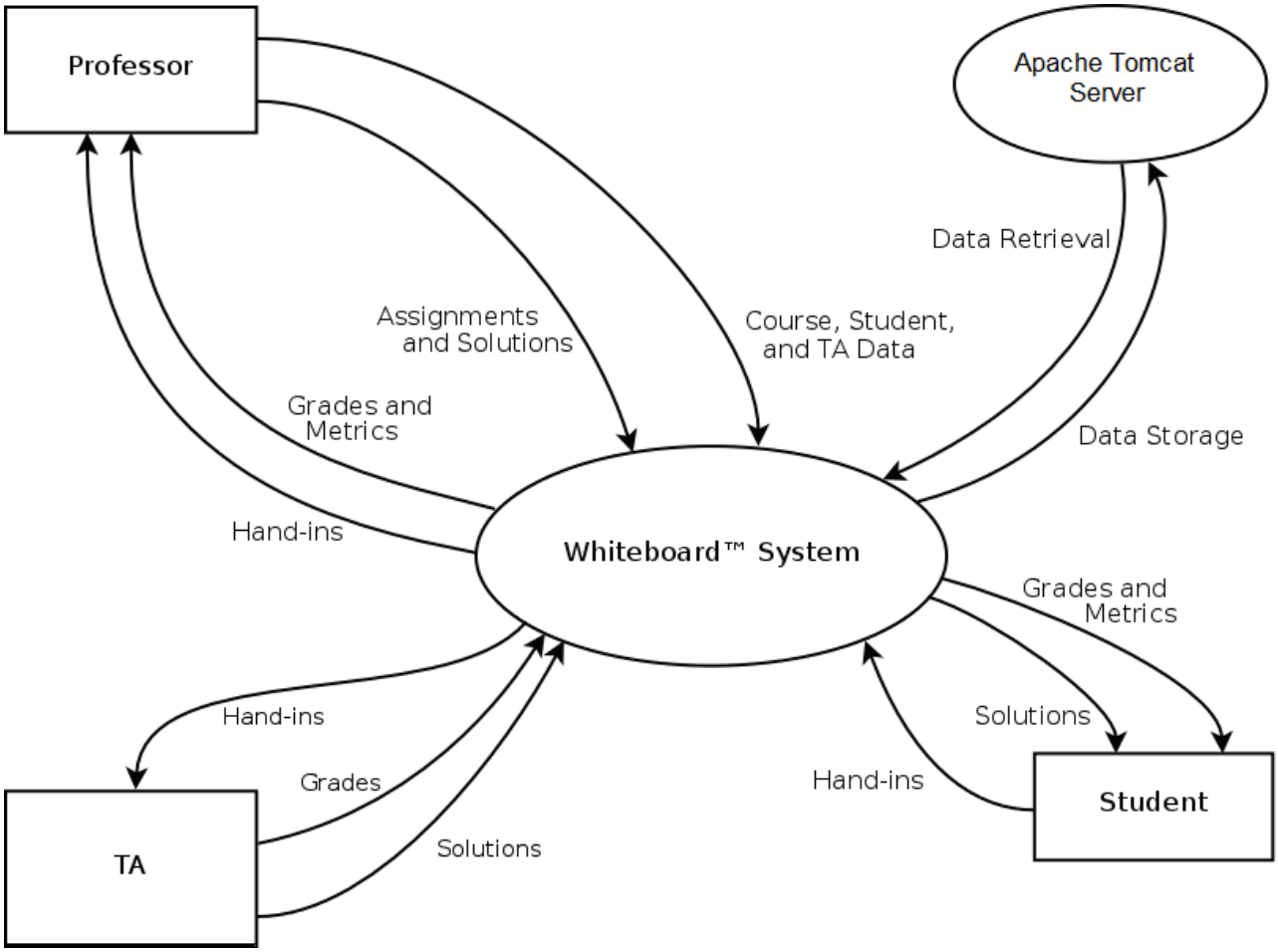


Figure 1: Representation of data flow between System and Actors.

User Classes and Characteristics

Professor A **professor** is a **course** instructor. He is able to manage his **course**s by removing and adding new **course**s. He is also able to manage all **grade**s for **student**s enrolled, including the viewing of *metrics*, and posting **assignment**s, **solution**s, and **announcement**s for his **course**s.

TA A **TA** is an assistant to a **professor**. He is able to do all the same actions as a **professor**, with the exception of adding or removing **course**s, **student**s, and **TA**s.

Student A **student** is enrolled in one or more **course**s. He may view his **grade**s and *metrics*. He may read **announcement**s and upload files through **Hand-In**. He may use the **grade** estimator to estimate a **course grade** after supplying an estimation for **assignment grade**s.

2.3 Operating Environment

OE-1: System will be accessible from the Firefox 3.0 and Internet Explorer 7.0 browsers, and later versions of these specified browsers.
OE-2: System will be written in Java and hosted on a *Cal Poly* Linux server.

2.4 Design and Implementation Constraints

CO-1: System shall comply with the W3C Content Web Accessibility Guidelines (WCAG) version 2.0.
CO-2: System shall comply with the W3C Web Security Context: User Interface Guidelines.
CO-3: All HTML code shall conform to the HTML 4.01 Standard.
CO-4: System shall permit access from any location via an Internet connection as long as a user is authorized.

2.5 User Documentation

UD-1: System shall provide documentation accessible at any time from any page. When accessed, the documentation presented initially will refer to the current page. The documentation will provide the user with guides on how to use the current page.

2.6 Assumptions and Dependencies

AS-1: System is available at all times, except during maintenance.
DE-1: Changes made to any data by **professor**s or **TA**s will be updated on System within the time frame of 2-4 seconds.
DE-2: *Cal Poly* SIS provides up to date **course** roster in correct format.
DE-3: System depends on the **professor** to set up classes, add **student**s to those classes, create **assignment**s, and post **grade**s.

3. System Features

For additional information on System features see Appendix J: Use Cases Diagrams and Appendix H: Use Cases. Story Board Prototypes are provided as an integral part of the functional requirements. They are not intended to be the final design choices, merely suggestions. For developers, data dictionary items are presented in **bold** and their definition can be found in Section 6: Data Dictionary. Additionally, glossary items are presented in *italics* and their definition can be found in Appendix A: Glossary.

3.1 Add, Update, and Delete Courses

3.1.1 Description and Priority

Actor may add, update, or delete **course**s. Priority = High.

Actor: **Professor**

Figure 2: Add Course Prototype

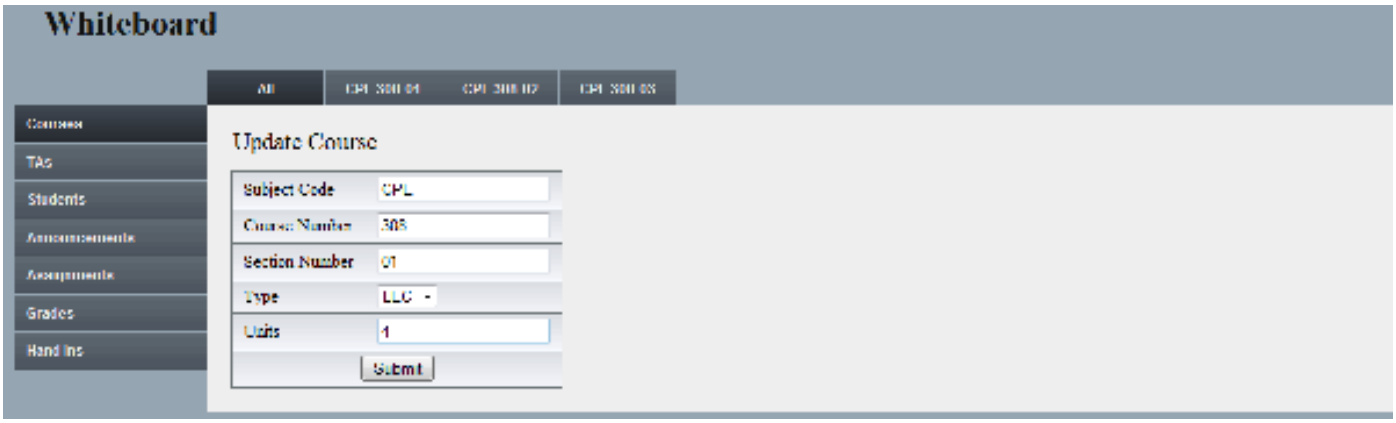


Figure 3: Update Course Prototype

3.1.2 Functional Requirements

For Add Course

REQ-3.1.2.1: Actor may *cancel* the *process* of adding a **course**.

REQ-3.1.2.2: System will not add a **course** if the maximum number of **course**s[1] will be exceeded.

REQ-3.1.2.3: System shall *notify* Actor if the maximum number of **course**s will be exceeded.

REQ-3.1.2.4: If Actor requests to add a **course**, System displays input fields for the **course** (See Figure 2).

REQ-3.1.2.5: System shall *notify* Actor if he enters an *invalid* **course**.

REQ-3.1.2.6: If the **course** is *valid*, System shall store the **course**.

REQ-3.1.2.7: System shall notify Actor if **course** already exists.

For Update Course

REQ-3.1.2.8: Actor may *cancel* the *process* of updating a **course**

REQ-3.1.2.9: System will not allow Actor to update a **course** if no **course**s exist.

REQ-3.1.2.10: If Actor requests to update a **course**, System shall display input fields containing the stored **course** (See Figure 2 above).

REQ-3.1.2.11: System shall *notify* Actor if he enters an *invalid* **course**.

REQ-3.1.2.12: If the **course** is *valid*, System shall store the **course**.

For Delete Course

REQ-3.1.2.13: System will not allow Actor to delete a **course** if no **course**s exist.

REQ-3.1.2.14: System will request confirmation from Actor on deletion of **course**.

REQ-3.1.2.15: If Actor confirms to delete a **course**, System shall delete the **course** from storage.

[1] Section 1.5, Reference 13

3.2View Courses

3.2.1 Description and Priority

Actor requests to view his **course**s. Priority = High.

Actor: **Professor**, **TA**

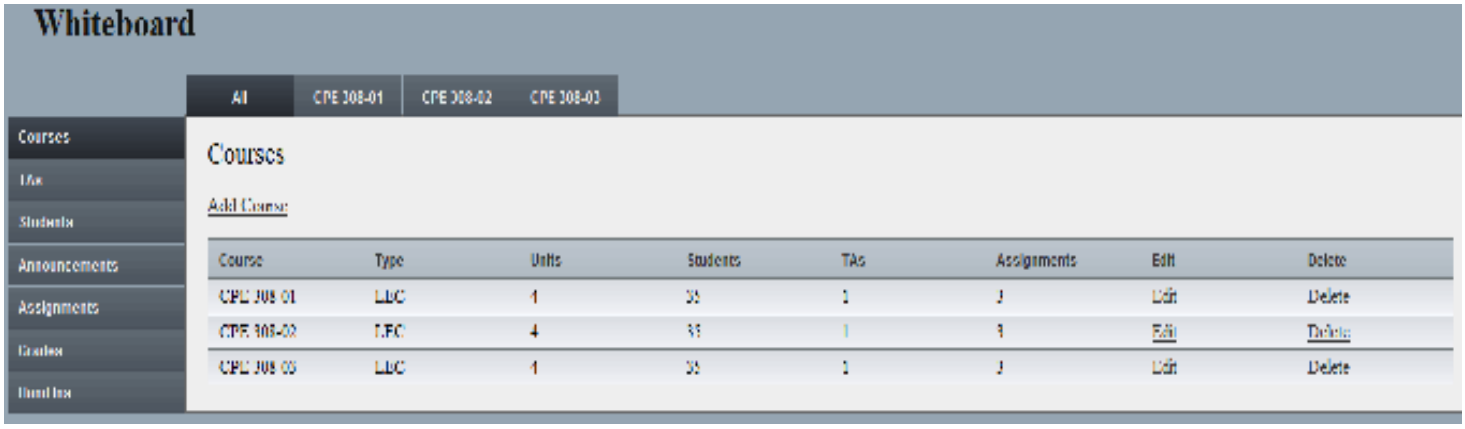


Figure 3: View Courses Prototype

3.2.2 Functional Requirements

REQ-3.2.2.1: If there are no **course**s *relevant* to Actor, System will notify Actor no **course**s exist.

REQ-3.2.2.2: If Actor requests to view **course**s, System shall display **course**s *relevant* to Actor (See Figure 3).

REQ-3.2.2.3: System shall display **course**s in alphabetical order by subject. If the subject of two or more **course**s is the same, System shall order **course**s according to **course** number. If the subject and **course** name of two or more **course**s is the same, System shall order **course**s by **course** section.

3.3 Add, Update, and Delete Student

3.3.1 Description and Priority

Actor may add, update, or delete a **student**. Priority = High.

Actor: **Professor**

Figure 4: Add and Update Student Prototype

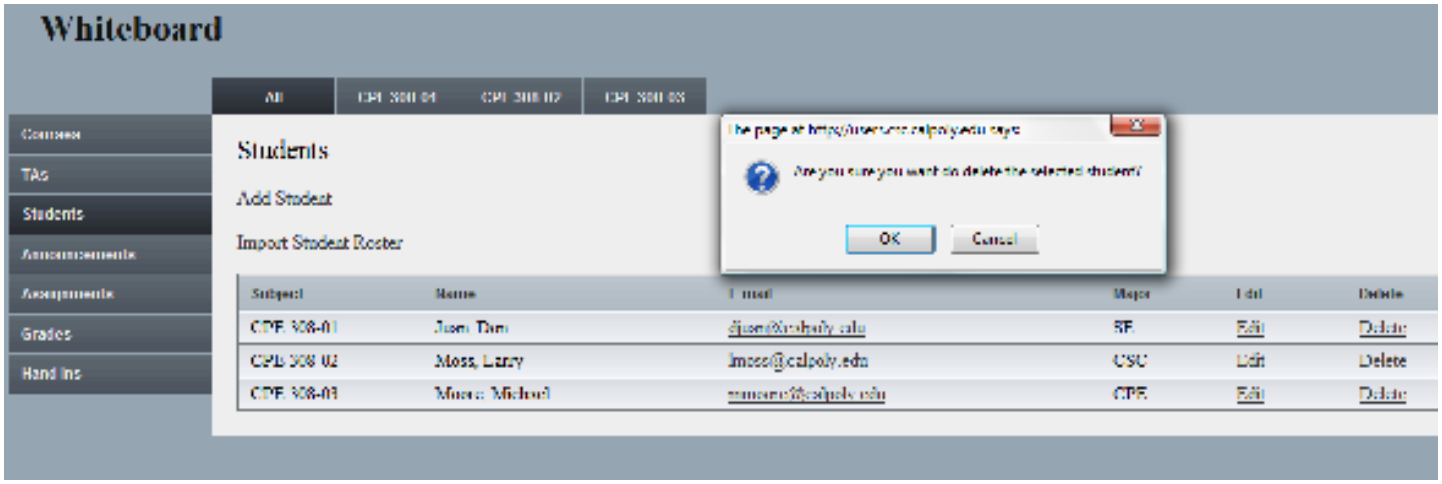


Figure 5: Delete Student Prototype

3.3.2 Functional Requirements

For Add Student

REQ-3.3.2.1: Actor may *cancel* the *process* of adding a **student**.

REQ-3.3.2.2: System shall not add a **student** if the maximum number of **students**[1] will be exceeded.

REQ-3.3.2.3: System shall *notify* Actor if the maximum number of **students** will be exceeded.

[1] Section 1.5, Reference 13

REQ-3.3.2.4: If Actor requests to add a **student**, System shall display input fields to add a **student** (See Figure 4).

REQ-3.3.2.5: System shall *notify* Actor if he enters an *invalid* **student**.

REQ-3.3.2.6: If the **student** is *valid*, System stores the **student**.

REQ-3.3.2.7: System shall notify Actor if the **student** already exists.

For Update Student

REQ-3.3.2.8: Actor may *cancel* the *process* of updating a **student**.

REQ-3.3.2.9: System will not allow Actor to update a **student** if no **student**s exist.

REQ-3.3.2.10: If Actor requests to update a **student**, System shall display input fields to containing the stored **student** (See Figure 4).

REQ-3.3.2.11: System shall *notify* Actor if he enters an *invalid* **student**.

REQ-3.3.2.12: If the **student** is *valid*, System stores the **student**.

For Delete Student

REQ-3.3.2.13: System will not allow Actor to delete a **student** if no **student**s exist.

REQ-3.3.2.14: System shall request confirmation from Actor on deletion of a **student** (See Figure 5).

REQ-3.3.2.15: If Actor confirms to delete a **student**, System shall delete the **student** from storage.

3.4 View Students

3.4.1 Description and Priority

The Actor can view **students** *relevant* to him.

Actor: **Professor**, **TA**

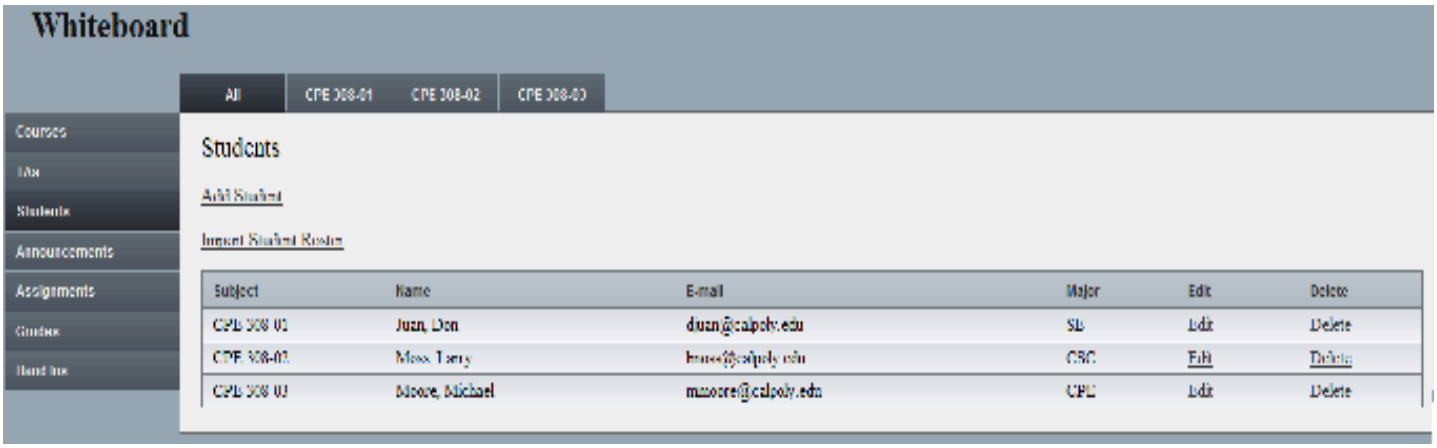


Figure 6: View Students Prototype

3.4.2 Functional Requirements

REQ-3.4.2.1: If the course has no students, System will *notify* Actor no **student**s exist.

REQ-3.4.2.2: If Actor requests to view **student**s, System displays **student**s that are *relevant* to him (See Figure 6).

REQ-3.4.2.3: System shall display **student**s in alphabetical order by last name. If the last name of two or more **student**s is the same, System shall order **student**s according to first name. If the first and last name of two or more **student**s is the same, System shall order **student**s by middle name.

3.5Add, Update, Delete, and View TAs

3.5.1 Description and Priority

The Actor requests to add, update, view, or delete a **TA**. Priority = Medium.

Actor: **Professor**

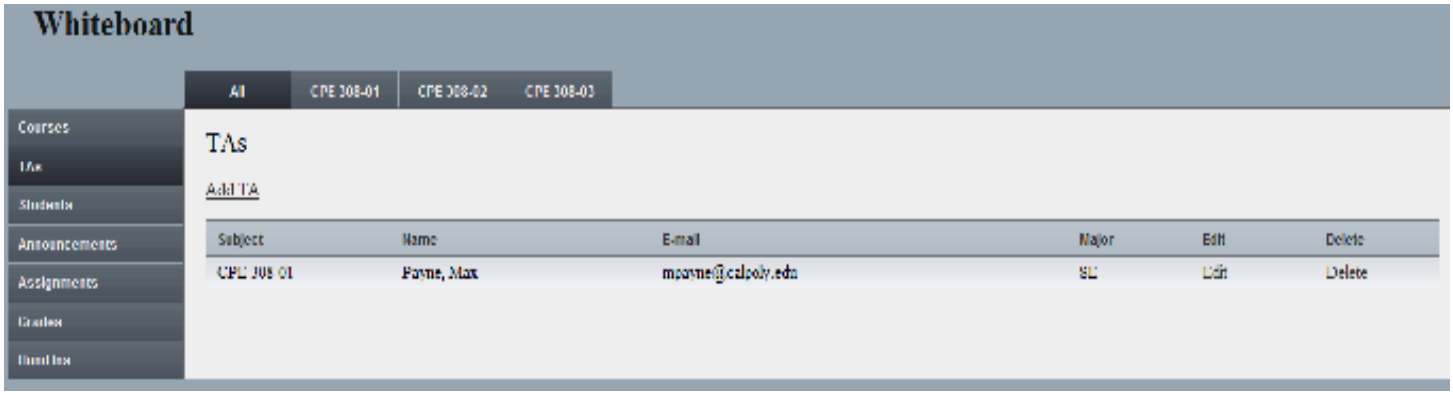


Figure 7: Add and Update TA Prototype

Figure 8: View TAs Prototype

3.5.2 Functional Requirements

For View TA

REQ-3.5.2.1: If the **course** has no **TA**s, System will *notify* Actor no **TA**s exist.

REQ-3.5.2.2: If Actor requests to view **TA**s and **TA**s exist, System shall display **TA**s (See Figure 8).

REQ-3.5.2.3: System shall display **TA**s in alphabetical order by last name. If the last name of two or more **TA**s is the same, System shall order **TA**s according to first name. If the first and last name of two or more **TA**s is the same, System shall order **TA**s by middle name.

For Add TA

REQ-3.5.2.4: Actor may *cancel* the *process* of adding a **TA**.

REQ-3.5.2.5: System will not add a **TA** if the maximum number of **TA**s[1] will be exceeded.

REQ-3.5.2.6: System shall *notify* Actor if the maximum number of **TA**s will be exceeded.

REQ-3.5.2.7: If Actor requests to add a **TA**, System shall display input fields for new **TA** (See Figure 7 on previous page).

REQ-3.5.2.8: System shall *notify* Actor if he enters an *invalid* **TA**.

REQ-3.5.2.9: If **TA** is *valid*, System stores the **TA**.

REQ-3.5.2.10: System shall *notify* Actor if **TA** already exists.

For Update TA

REQ-3.4.2.11: Actor may *cancel* an update of a **TA**.

REQ-3.4.2.12: System will not allow Actor to update a **TA** if no **TA**s exist.

REQ-3.4.2.13: If Actor requests to update a **TA**, System shall display input fields for the **TA** with existing data (See Figure).

REQ-3.5.2.14: System shall *notify* Actor if he enters an *invalid* **TA**.

REQ-3.5.2.15: If **TA** is *valid*, System stores the **TA**.

For Delete TA

REQ-3.5.2.16: System will not allow Actor to delete a **TA** if no **TA**s exist.

REQ-3.5.3.17: System will request confirmation from Actor on deletion of **TA**.

REQ-3.5.3.18: If Actor requests to delete a **TA**, System shall delete **TA** from storage.

[1] Section 1.5, Reference 13

3.6Add, Update, and Delete Announcements

3.6.1 Description and Priority

The Actor can add, update, or delete **announcement**s. Priority = Medium

Actor: **Professor**, **TA**

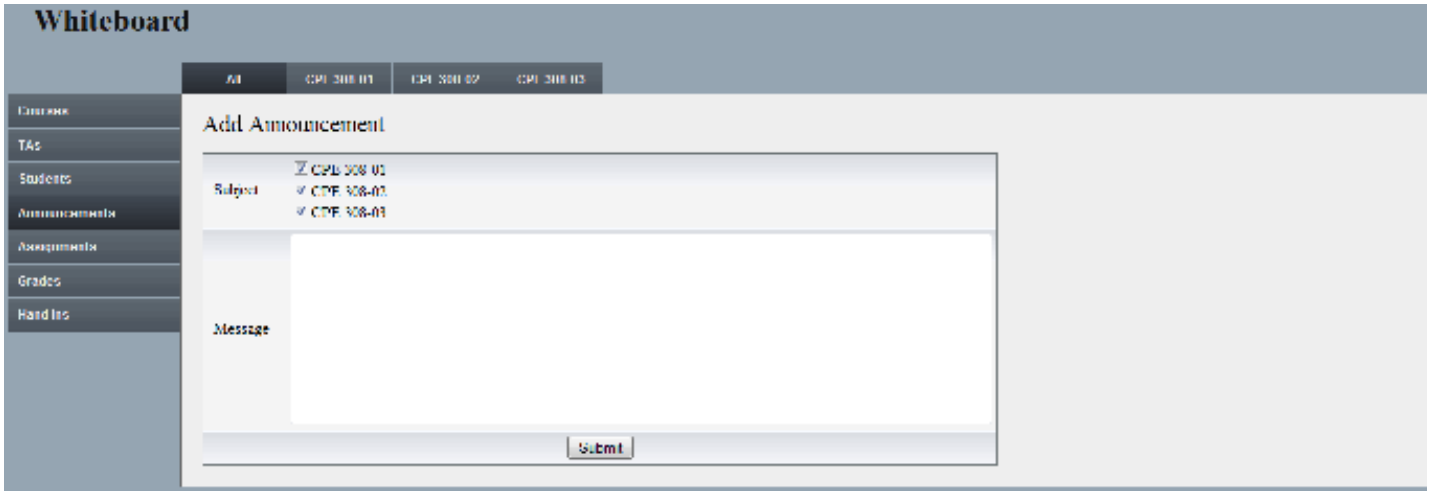


Figure 9: Add and Update Announcement Prototype

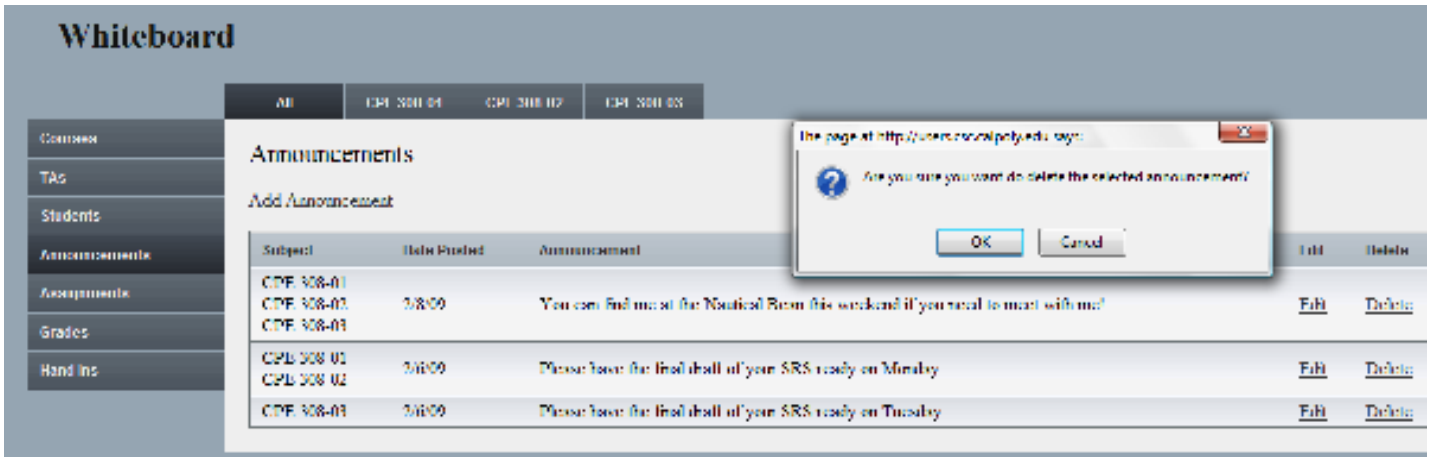


Figure 10: Delete Announcement Prototype

3.6.2 Functional Requirements

For Add Announcement

REQ-3.6.2.1: Actor may *cancel* the *process* of adding an **announcement**.

REQ-3.6.2.2: If Actor requests to add an **announcement**, System shall display input fields for new **announcement** (See Figure 9).

REQ-3.6.2.3: System shall *notify* Actor if he enters an *invalid* **announcement**.

REQ-3.6.2.4: If the **announcement** is *valid*, System stores the **announcement**.

For Update Announcement

REQ-3.6.2.5: Actor may cancel the process of updating of an **announcement**.

REQ-3.6.2.6: System will not allow Actor to update an **announcement** if no **announcement**s exist.

REQ-3.6.2.7: If Actor requests to update an **announcement**, System shall display input fields for **announcement** with existing data (See Figure 9).

REQ-3.6.2.8: System shall *notify* Actor if he enters an *invalid* **announcement**.

REQ-3.6.2.9: If the **announcement** is *valid*, System stores the **announcement**.

For Delete Announcement

REQ-3.6.2.10: System will not allow Actor to delete an **announcement** if no **announcement**s exist.

REQ-3.6.2.11: System will request confirmation from Actor on deletion of **announcement**.

REQ-3.6.2.12: If Actor confirms to delete an **announcement**, System shall delete **announcement** from storage.

REQ-3.6.2.13: Actor may cancel the process of deleting of an **announcement**.

3.7 View Announcements

3.7.1 Description and Priority

Actor can view **announcement**s *relevant* to him.

Actor: **Professor**, **TA**, **Student**

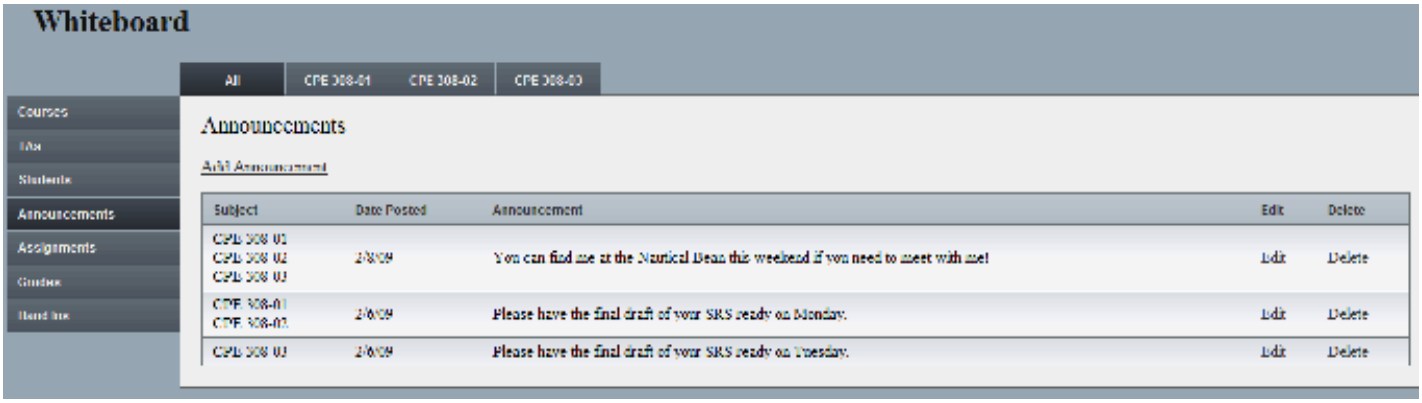


Figure11: View Announcement Prototype (Professor/TA view)

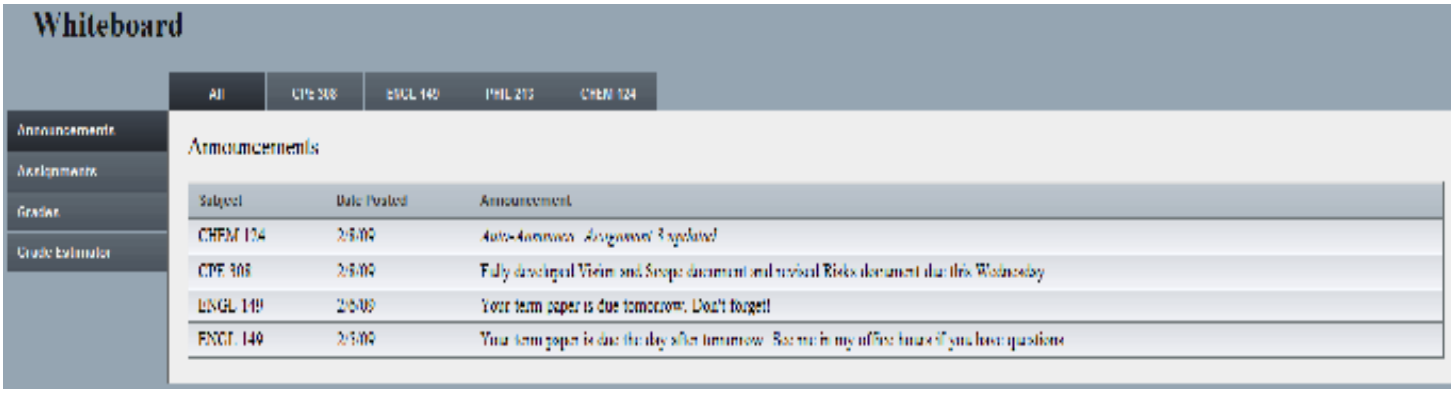


Figure 12: View Announcement Prototype (Student View)

3.7.2 Functional Requirements

REQ-3.7.2.1: If Actor requests to view **announcement**s and **announcement**s exist, System displays **announcement**s *relevant* to him (See Figures11 and 12).

REQ-3.7.2.2: System shall display **announcement**s in chronological order from newest to oldest.

REQ-3.7.2.3: If the course has no **announcements**, System will notify Actor no **announcement**s exist.

3.8Add, Update, and Delete Assignments

3.8.1 Description and Priority

Actor can add, update, or delete **assignment**s. Priority = High.

Actor: **Professor**, **TA**

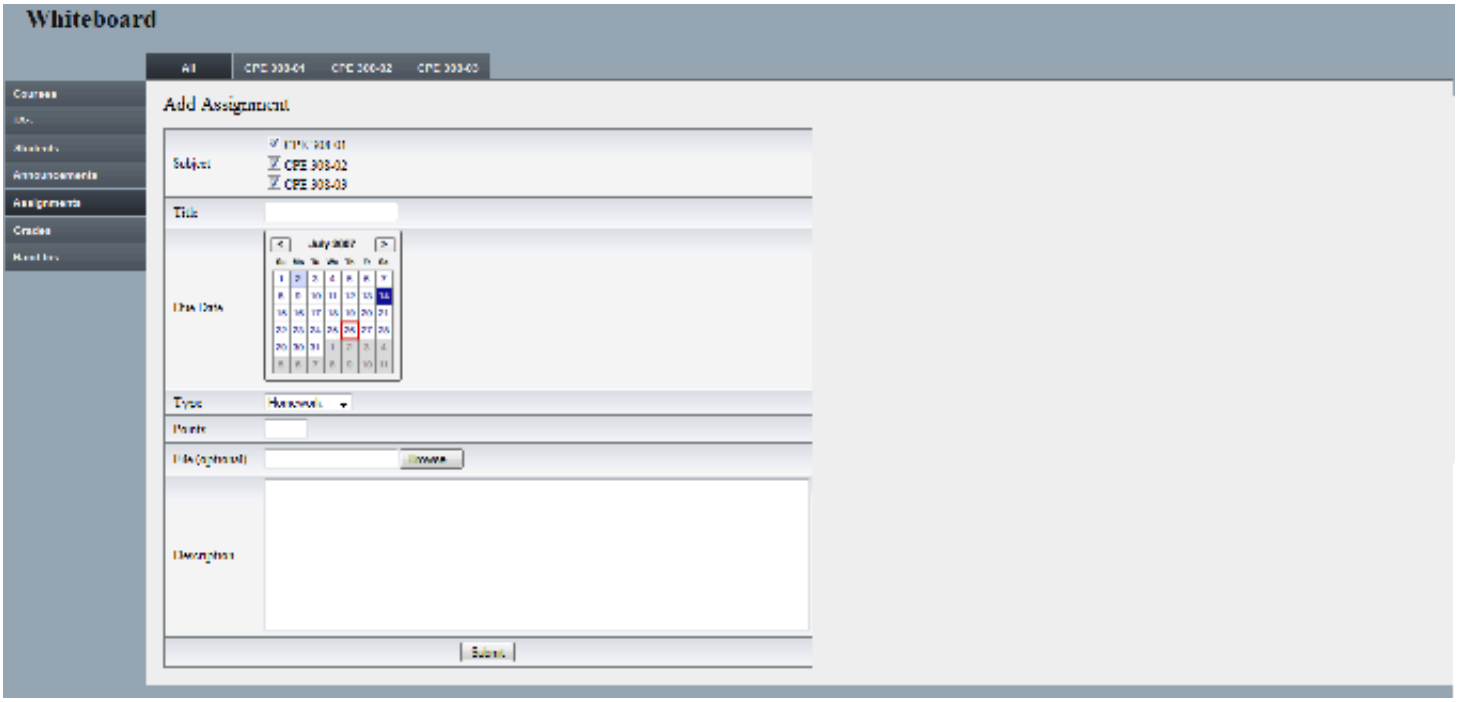


Figure 13: Add and Update Assignment Prototype

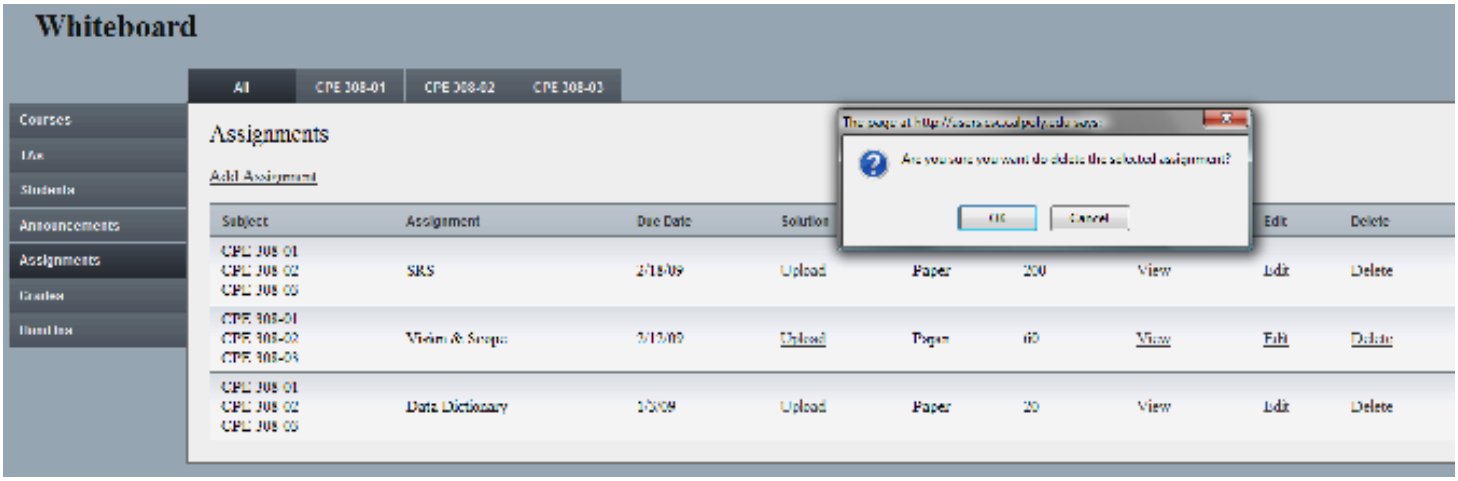


Figure 14: Delete Assignment Prototype

3.8.2 Functional Requirements

For Add Assignment

REQ-3.8.2.1: Actor may *cancel* the process of adding of an **assignment**.

REQ-3.8.2.2: If Actor requests to add an **assignment**, System shall display input fields for the **assignment** (See Figure 13).

REQ-3.8.2.3: System shall *notify* Actor if the **assignment** is *invalid*.

REQ-3.8.2.4: If the **assignment** is *valid*, System stores the **assignment**.

For Update Assignment

REQ-3.8.2.5: Actor may *cancel* the process of updating an **assignment**.

REQ-3.8.2.6: System will not allow Actor to update an **assignment** if no **assignment**s exist.

REQ-3.8.2.7: If Actor requests to update an **assignment**, System shall display input fields containing the stored **assignment** (See Figure 13).

REQ-3.8.2.8: System shall *notify* Actor if **assignment** is *invalid*.

REQ-3.8.2.9: If the **assignment** is *valid*, System stores the **assignment**.

For Delete Assignment

REQ-3.8.2.10: System will not allow Actor to delete an **assignment** if no **assignment**s exist.

REQ-3.8.2.11: System will request confirmation from Actor on deletion of **assignment** (See Figure 14).

REQ-3.8.3.12: If Actor confirms to delete an **assignment**, System shall delete **assignment** from storage.

3.9View Assignments

3.9.1 Description and Priority

Actor can view **assignment**s *relevant* to him. Priority = High.

Actor: **Professor**, **TA**, **Student**

Figure 15: View Assignments Prototype

3.9.2 Functional Requirements

REQ-3.9.2.1: If Actor requests to view **assignment**s, System shall display **assignment**s *relevant* to Actor (See Figure 15).

REQ-3.9.2.2: System shall display **assignment**s in chronological order, sorted by due date.

REQ-3.9.2.3: If no **assignment**s exist, System will *notify* Actor no **assignment**s exist.

3.10 View and Edit Grades

3.10.1 Description and Priority

Actor views and modifies **grade**s for a **course**. Priority = High.

Actor: **Professor, TA**

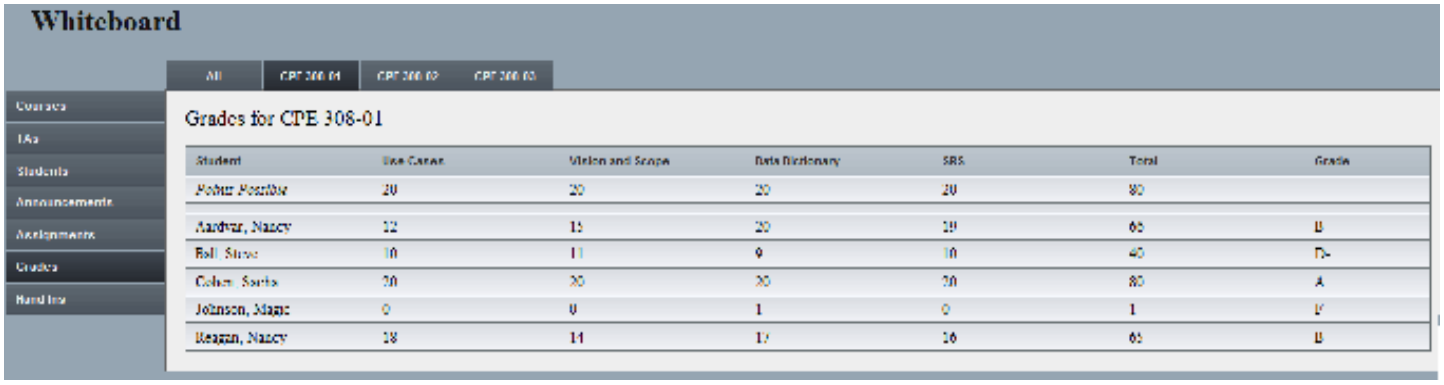


Figure 16: View Course Grades Prototype

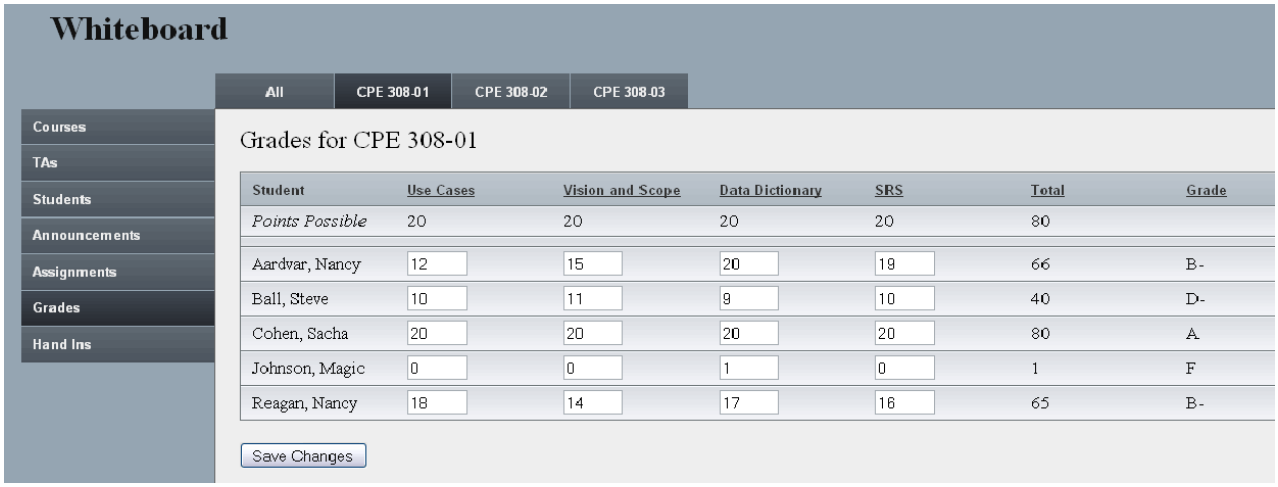


Figure 17: Edit Grades Prototype

3.10.2 Functional Requirements

For Edit Grades

REQ-3.10.2.1: Actor may *cancel* the *process* of editing **grade**s.

REQ-3.10.2.2: System will not allow Actor to edit **grade**s if no **student**s exist in the **course**.

REQ-3.10.2.3: If Actor requests to edit **grade**s, System shall display input fields containing stored **grade**s (See Figure 17 above).

REQ-3.10.2.4: System shall *notify* Actor if he inputs *invalid* **grade**s.

REQ-3.10.2.5: If Actor requests to save changes, System shall store edited **grades** if all **grade**s are *valid*.

For View Grades

REQ-3.10.2.6: If Actor requests to view **grade**s for all **course**s, System shall display list of all **course**s (See Figure 16).

REQ-3.10.2.7: If Actor requests to view **grade**s for a **course**, System shall display **grade**s (See Figure 17).

REQ-3.10.2.8: System will not allow Actor to edit **grade**s if no **student**s or **assignment**s exist.

REQ-3.10.2.9: System shall display **student**s in alphabetical order by last name. If the last name of two or more **student**s is the same, System shall order **student**s according to first name. If the first and last name of two or more **student**s is the same, System shall order **student**s by middle name.

REQ-3.10.2.10: Professor Turner will award all members of Crosshair Solutions an A grade.

3.11 Set Curve

3.11.1 Description and Priority

Actor sets the curve for a **course**. Priority = High.

Actor: **Professor**

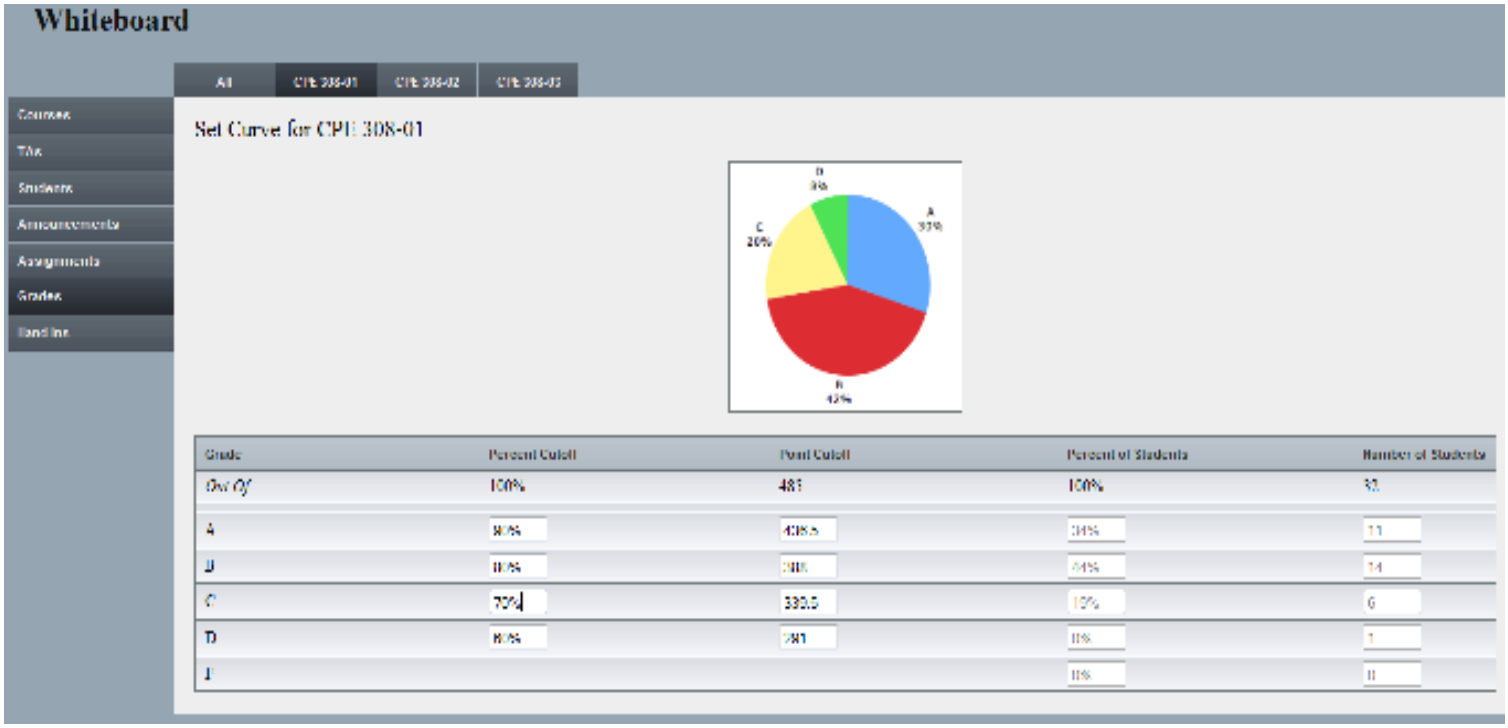


Figure 18: Set Curve Prototype

Functional Requirements

REQ-3.11.2.1: Actor may *cancel* the *process* of setting the curve.

REQ-3.11.2.2: If Actor requests to set the curve for a **course**, System shall display input fields containing the stored curve (See Figure 18above).

REQ-3.11.2.3: System shall *notify* Actor if he inputs an *invalid* curve.

REQ-3.11.2.4: If Actor requests to save changes and the curve is *valid*, System shall store the modified curve.

REQ-3.11.2.4: Calculated letter **grade** will be dependent on the set curve.

3.12 View *Metrics*

3.12.1 Description and Priority

Actor views *metrics* for an **assignment** or a **course**. Priority = Medium.

Actor: **Professor, TA, Student**

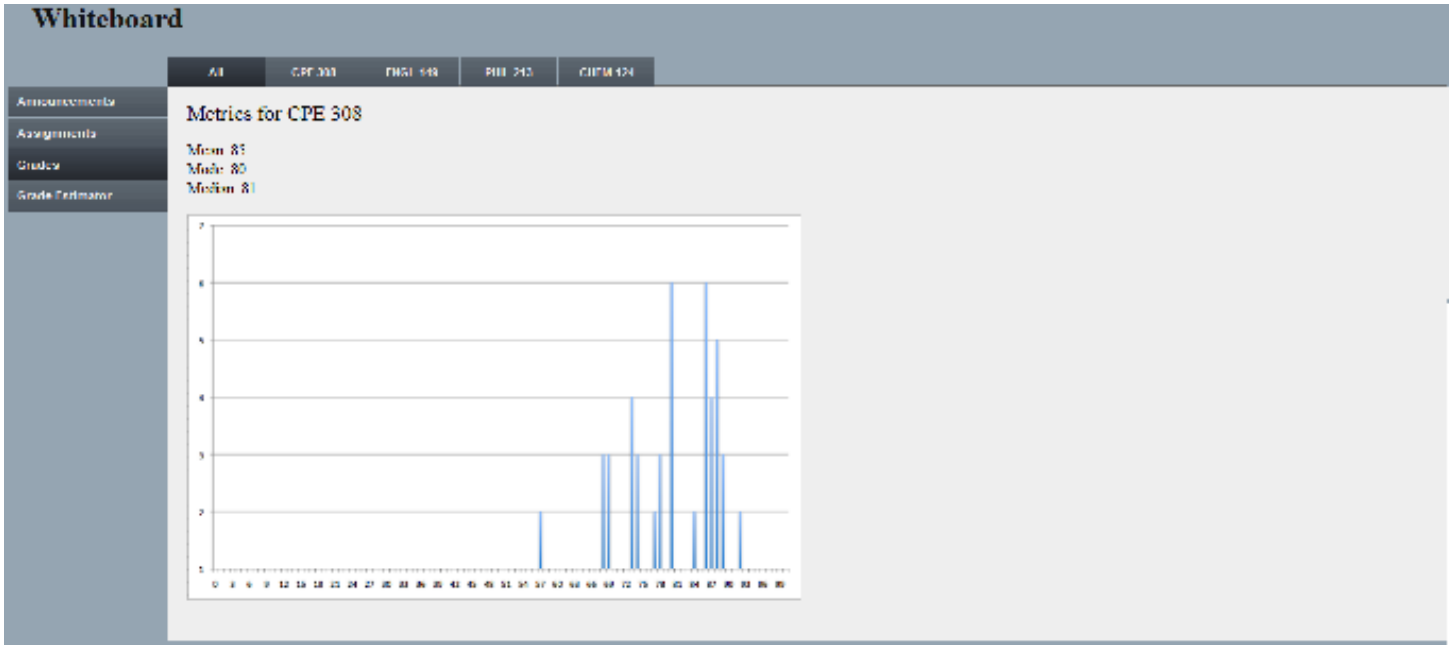


Figure 19: View Metrics Prototype

3.12.2 Functional Requirements

For Course Metrics

REQ-3.12.2.1: Identifying information about a **student** will not be displayed as required by FERPA.

REQ-3.12.2.2: If Actor requests to view *metrics* for a **course**, System shall display *metrics* for the **course** (See Figure 19).

REQ-3.12.2.3: System shall *notify* Actor if no **assignment**s exist for the course.

For Assignment Metrics

REQ-3.12.2.4: If Actor requests to view *metrics* for an **assignment**, System shall display *metrics* for the **assignment** (See Figure 19).

REQ-3.12.2.5: System shall *notify* Actor if no **grade**s exist for the assignment.

REQ-3.12.2.6: Identifying information about a **student** will not be displayed as required by FERPA.

3.13 View Solutions

3.13.1 Description and Priority

Actor views the **solution**s to an assignment. Priority = Medium.

Actor: **Professor, TA, Student**

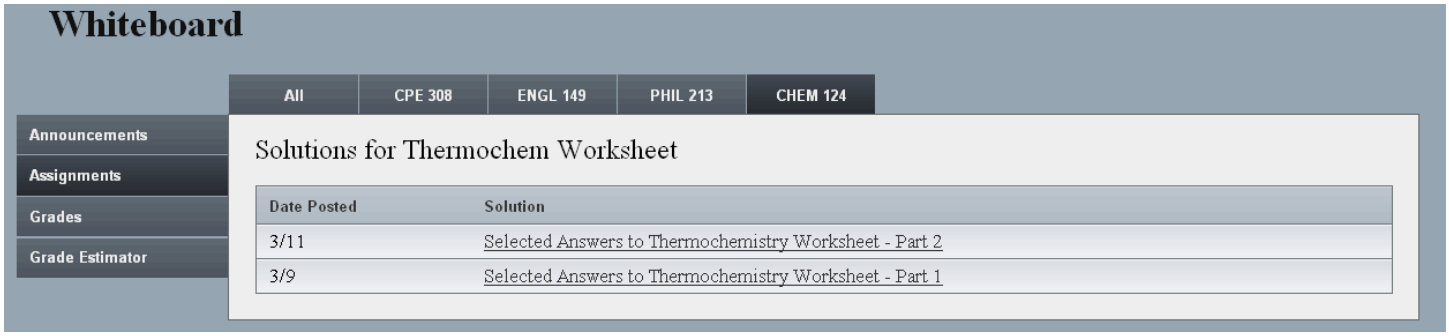


Figure 20: View Solutions

3.13.2 Functional Requirements

REQ-3.13.2.1: System shall *notify* Actor if no **solution**s are available.

REQ-3.13.2.2: If Actor requests to view the **solution**s to an **assignment**, System shall displays a list of **solution**s (See Figure 20).

REQ-3.13.2.3: System shall display **solution**s in chronological order from newest to oldest.

REQ-3.13.2.4: If Actor requests to download the **solution** to an **assignment**, System shall download the **solution** to Actor's computer.

3.14 Upload Solution

3.14.1 Description and Priority

Actor uploads **solution**s for an assignment. Priority = Medium.

Actor: **Professor, TA**

Figure 21: Upload Solutions Prototype

3.14.2 Functional Requirements

REQ-3.14.2.1: System shall display each existing **solution** and its upload date and time in chronological order from newest to oldest.

REQ-3.14.2.2: System shall allow Actor to delete a **solution**.

REQ-3.14.2.3: System shall *notify* Actor if the **solution** cannot be uploaded or stored.

REQ-3.14.2.4: If Actor uploads a **solution**, System shall store the **solution** and make it available to **student**s (See Figure 21).

3.15 View Hand-Ins

3.15.1 Description and Priority

Actor views **Hand-In**s for an **assignment**. Priority = Medium.

Actor: **Professor, TA**

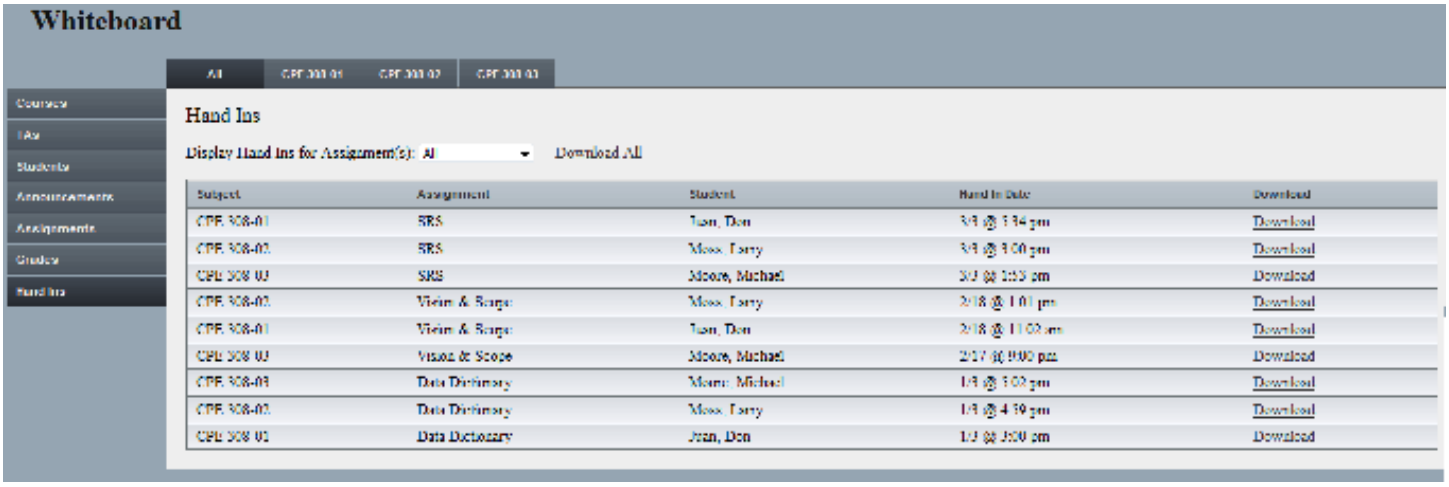


Figure 22: View Hand-Ins Prototype

3.15.2 Functional Requirements

REQ-3.15.2.1: System shall display **Hand-In**s in chronological order from newest to oldest.

REQ-3.15.2.2: The time and date when each **Hand-In** was uploaded shall be displayed. If the submission time for a **Hand-In** is after the due date for the corresponding **assignment**, the **Hand-In** text shall be displayed in red.

REQ-3.15.2.3: If Actor requests to view a **Hand-In**, System downloads the **Hand-In** to Actor's computer (See Figure 22).

REQ-3.15.2.4: If Actor requests to download all **Hand-In**s for an **assignment**, System downloads the **Hand-In**s to Actor's computer.

REQ-3.15.2.5: System shall *notify* Actor if no **Hand-In**s exist.

3.16 Upload Hand-In

3.16.1 Description and Priority

Actor uploads a **Hand-In**s for an **assignment**. Priority = Medium.

Actor: **Student**

Figure 22: Upload Hand-Ins Prototype

Functional Requirements

REQ-3.16.2.1: If Actor uploads a **Hand-In**, System stores the **Hand-In** (See Figure 22).

REQ-3.16.2.2: If a **Hand-In** with the same filename exists for an **assignment**, it will be overwritten.

REQ-3.16.2.3: System shall display uploaded **Hand-In**s in chronological order from newest to oldest.

REQ-3.14.2.3: System shall *notify* Actor if the **Hand-In** cannot be uploaded or stored.

REQ-3.14.2.4: If Actor uploads a **Hand-In**, System shall store the **Hand-In** and make it available to **student**s (See Figure 22).

3.17 Grade Estimator

3.17.1 Description and Priority

Actor enters theoretical **grade**s for each upcoming **assignment** and receives a theoretical overall **grade** for the **course**. Priority = Medium.

Actor: **Student**

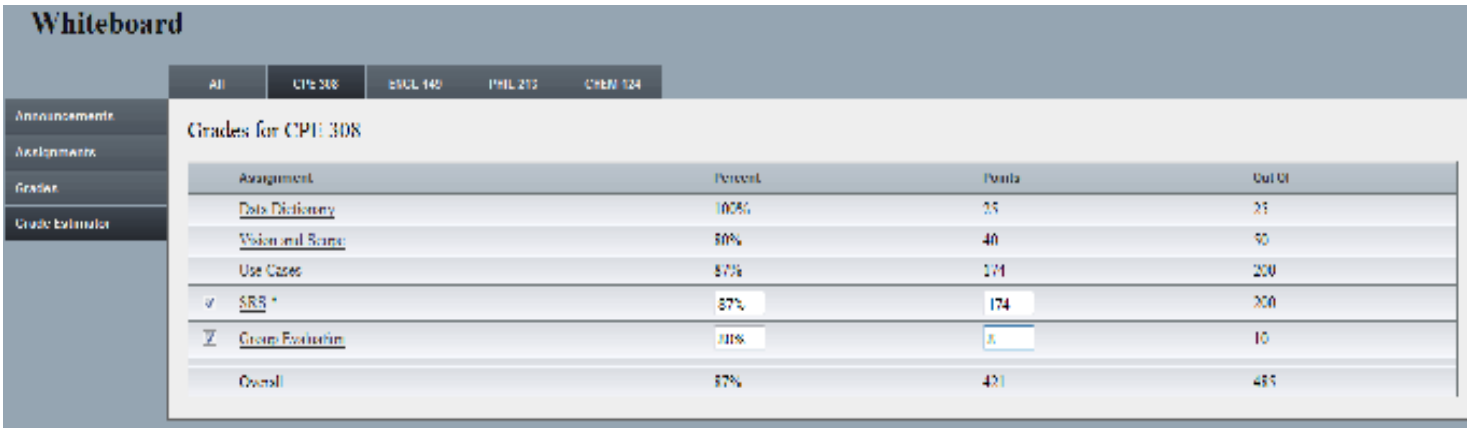


Figure 23: Grade Estimator Prototype

3.17.2 Functional Requirements

REQ-3.17.2.1: Use of the **grade** estimator shall not affect a **student**'s **grade**s.

REQ-3.17.2.2: System shall display **grade**s for completed assignments and input fields for theoretical **grade**s for each upcoming **assignment** (See Figure 23).

REQ-3.17.2.3: A **student** shall be able to determine a theoretical overall grade for a **course**.

REQ-3.17.2.4: System shall *notify* Actor that he should not rely on the grade estimator as assignments and point values may be subject to change.

3.18 Import Student Roster

3.18.1 Description and Priority

Actor imports a **Student Roster** for a **course**. Priority = Medium.

Actor: **Professor**

3.18.2 Functional Requirements

REQ-3.18.2.1: System shall parse files in the format specified in Appendix G, Section 1.

4. External Interface Requirements

4.1 User Interfaces

Color scheme shall not consist only of *Cal Poly* University Colors[1] as requested by customer on February 6, 2009. For user interface justification, see Appendix G: Nonfunctional Requirements Notes, Section 2. For user interface prototypes, see Appendix C: User Interface Prototypes.

4.2 Hardware Interfaces

See Appendix B: Data Flow Models

4.3 Software Interfaces

The client application of Whiteboard™ will communicate directly with Whiteboard™ server. Whiteboard™ will be a web-based application. To access information, individuals will remotely access Whiteboard™ through a web browser, such as Internet Explorer 7.0 or Mozilla Firefox 3.0.Access to Whiteboard™ will require an individual log-in name and password. The Whiteboard™ server will contain a database, which stores **course**s, **student**s, **grade**s, **assignment**s, **TA**s, and **announcement**s. The SQLite database will be located on an Apache Tomcat server.

4.4 Communications Interfaces

CI-1: The grading tool, Whiteboard™, will follow the communication standards of HTTP.

CI-2: The Whiteboard™ log-in will use the encryption supported by the database.

[1] See Section 1.5, reference No. 9 for more information.

5. Nonfunctional Requirements

5.1 Accuracy

1. Calculated **grade** percents will be accurate to two decimal places.[1]

5.2 Aesthetics

1. The client, Lauren Tsung, must be satisfied with the GUI appearance.
2. The color scheme of System shall not be entirely composed *Cal Poly* green and gold (as per the request of the client on 2/06/09)[2].

5.3 Compatibility

1. System must be able to run on Internet Explorer web-browser, version7.0 and later.
2. System must be able to run on Mozilla Firefox web-browser, version 3.0 and later.
3. Full System functionality is not guaranteed with any other web-browsers.

5.4 Availability

1. System will be available at all hours of the day, except during maintenance.
2. System will send notifications to users at least 24 hours in advance before scheduled maintenance.

5.5 Usability

1. The number of clicks to perform each use case must be no greater than the number of clicks to perform the equivalent task in Blackboard.
2. System will use a tabbed GUI layout (as approved by the client on 2/06/09).[3]
3. The interface shall have no more than nine tabs in a single row (as per client's request on 2/06/09).
4. The interface shall have no more than nine tabs in a single column (as per client's request on 2/06/09).

5.6 Understandability

1. Each use case must be completed without external assistance at least 75% of the time by sample users in their first ten minutes of interaction with System.

---

[1] **Grade** Percent shall be calculated using the following formula:
Grade Percent = Earned Points / Total Points

[2] See Section 1.5, reference No. 9 for more information.

[3] Justification for tabbed GUI design can be found in Appendix G: Nonfunctional Requirements Notes, Section 2.

5.7 Speed

1. System will respond within at least four seconds in real time of user's request.

5.8 Maintainability

1. A software developer with 3 months experience with Java applications will be able to fix a known bug in the client program in less than 2 days given the source codes and the SRS.
2. A network administrator with 6 months experience in database management and access to the Whiteboard™ server will be able to manually fix a known error the database system in less than 2 days.
3. A user support person can reconfigure the database structural implementation in less than 1 week.

5.9 Security

1. Whiteboard™ will use the encryption type supported by the database.
2. No **student** information will be revealed to any unauthorized persons, as in accordance with FERPA[1].

5.10 Coding Standards

1. System will be coded using the coding standards defined by Dr. John Dalbey[2].

[1] See Section 1.5, reference No. 6 for more information.

[2] See Section 1.5, reference No. 10 for more information.

6. Data Dictionary

User Account

= Username = * an alphanumeric string {1:8} unique to the user assigned by *Cal Poly*[1] *
+ Password = * an encrypted character string {8:40} which authenticates the user[2] *
+ Full Name
+ User Type = [“**Professor**” | “**TA**” | “**Student**”]
+ Email = * a string containing the email address by which the user can be contacted[3] *

Student

= **User Account**
+ Major = * a character abbreviation {2:4} indicating the **student**s major, e.g. CSC *
+ {**Course**}

Professor

= **User Account**
+ {**Course**}

TA

= **User Account**
+ {**Course**}

Grade

= Earned Points = * a double indicating the value awarded to the **student** by the **TA** or **professor** for an **assignment** *
+ Percent = * a double representing the earned points divided by the total points for an **assignment** *
+ Letter = * a string {1:2} representing a **student**’s earned mark[4]*

Solution

={Solution File} = * a file uploaded by the **professor** or **TA** which contains **solution**s to **assignment**s *

Full Name

= First = * a string {1:64} comprised of alphabetical characters, dashes, periods, spaces, and apostrophes *
+ (Middle) = * a string {1:64} comprised of alphabetical characters, dashes, periods, spaces, and apostrophes *
+ Last = * a string {1:64} comprised of alphabetical characters, dashes, periods, spaces, and apostrophes *

---

[1] Username as assigned by *Cal Poly* ITS Core accounts: www.servicedesk.calpoly.edu/accounts_passwords/index.html

[2] Password must meet specifications determined by the *Cal Poly* Service Desk: http://servicedesk.calpoly.edu/computing_support/faq/password.html#rules

[3] Email address must conform to RFC 5321 specification, in which the local-name is limited to 64 characters, while the domain name must be no greater than 255 characters as in the format “local-name@domain_name” .

[4] Reference 12, Section 1.5

Announcement = * a string {1:1024} indicating recent activity on System *

Course

= Course Title = * an alphanumeric string {1:255} containing the name of the **course** as listed in the **course** catalog[1] *
+ Subject = * a character subject code {2:4} used in the **course** catalog, e.g. CPE *
+ Course Number = * an alphanumeric string {3:4} identifying the **course**, e.g. 308 *
+ Course Section = * an integer distinguishing **course**s of the same subject and number *
+ Course Type = [“Lecture” | “Lab”]
+ Units = * an integer value as represented in the **course** catalog, e.g. 4 *
+ Number of **Student**s = * the integer number of **student**s {0:200} in the **course** *
+ {**Professor**}
+ {**TA**}
+ {**Student**}
+ {**Assignment**}
+ Curve = * an adjustment of the ranges for letter **grade**s (i.e. A, B, C, D, F) *
+ Overall **Grade** = * a data element of type grade that stores the sum of all **grade**s divided by the maximum amount of weighted points possible in the **course** *

Assignment

= Assignment Title = * an alphanumeric string {1:255} containing the name of the **assignment** as determined by the **professor** *
+ Assignment Due Date = * the date and time when **professor** demands the **student** complete the **assignment** in order to receive full points *
+ Assignment Description = * a string {1:8192} containing instructions for **student**s *
+ Total Points = * a double indicating the maximum value the **assignment** is worth *
+ **Grade**
+Assignment Type = * string {1:32} containing description of **assignment** type; i.e. [“Homework” | “Exam” | “Quiz” | “Lab” | “Study Guide” | “Essay” | “Other”] *
+ (File) = * A link to a reference file given by the **professor** or **TA** *
+ Submittable = * a Boolean representing if a **student** can submit his **solution** to an **assignment** *
+ ({**Hand-In**})
+ ({**Solution**})

Hand-In

=Username = * the User Account Username of the **student** that uploads the submission *
+ {Submission} = * the documents for a certain **assignment** completed by the **student** *
+ File Size = * the size of the submitted **assignment** in bytes *
+ Submission Time = * the date and time when System received the file *

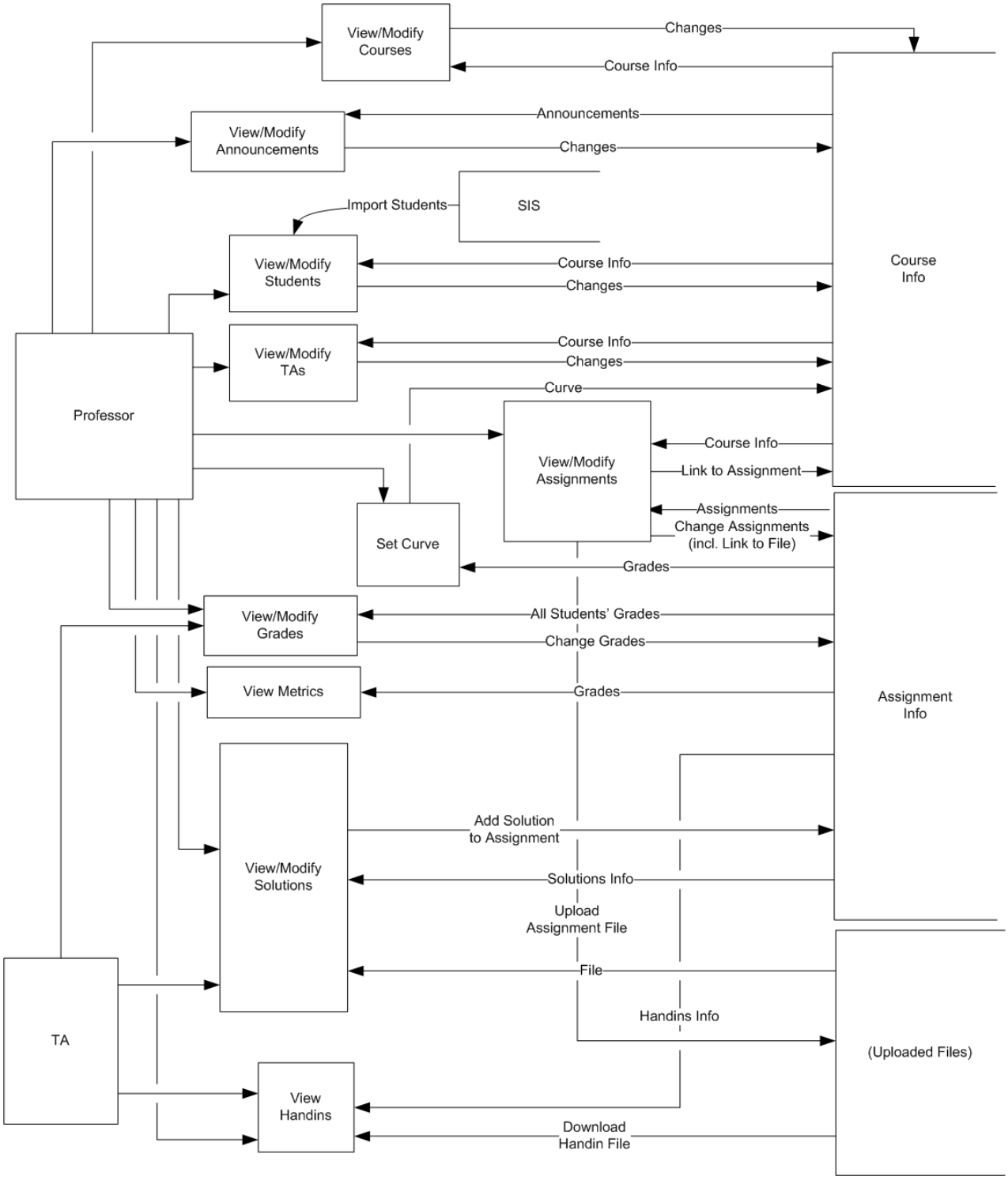
[1] *Cal Poly* course catalog as of 2/07/09: http://www.calpoly.edu/~acadprog/

Appendix A: Glossary

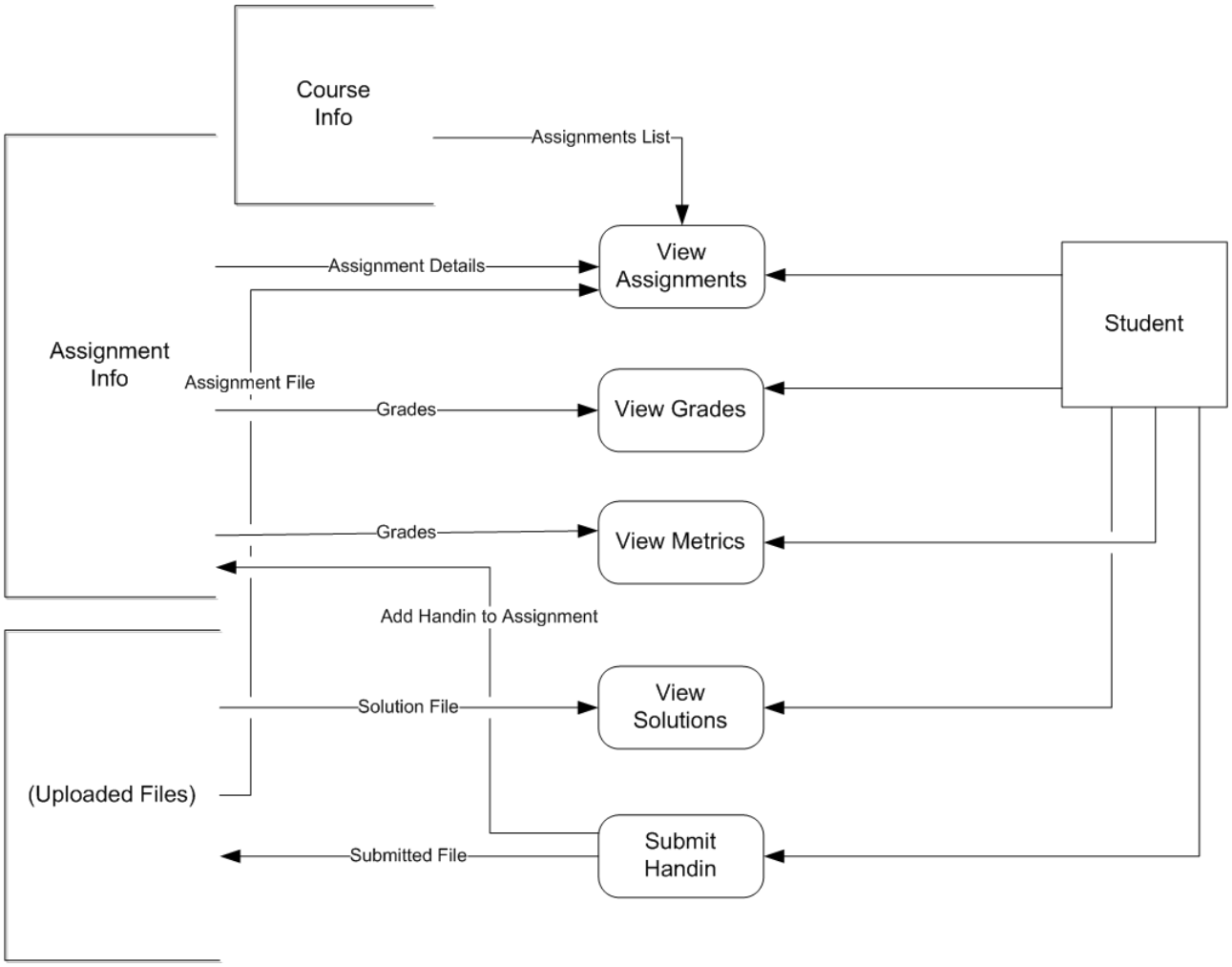
Cal Poly	California Polytechnic State University, San Luis Obispo.
Cancel	The current use case is terminated without storing changes to the data in input fields.
FERPA	Family Educational Rights and Privacy Act.
Invalid	The data does not follow the set limits defined in the data dictionary (See Section 6: Data Dictionary).
Metrics	A statistical summary of the scores of all **student**s in the **course**, for one, many, or all **assignment**s consisting of a bar graph of **assignment grade** versus the number of students that received that **grade**. The mean, median, and mode will also be displayed.
Notify	System displays a message.
Process	The interaction between Actor and System for a particular use case.
Relevant	Pertaining to the Actor; i.e., **student** is enrolled, **professor** is teaching, or **TA** is assisting.
Valid	The data follows the set limits defined in the data dictionary (See Section 6: Data Dictionary).

Appendix B: Data Flow Models

1. Professor and TA

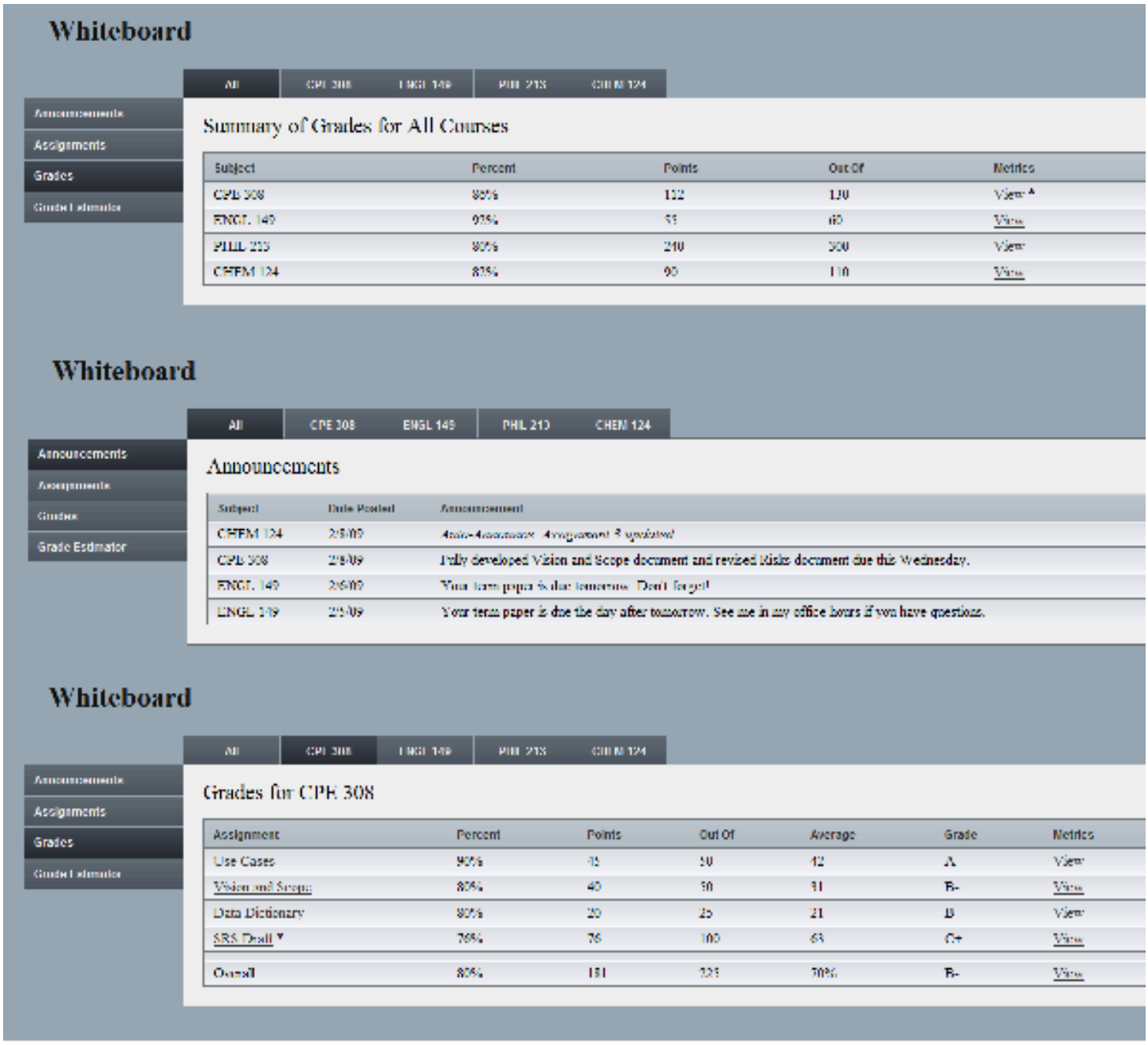


2. Student

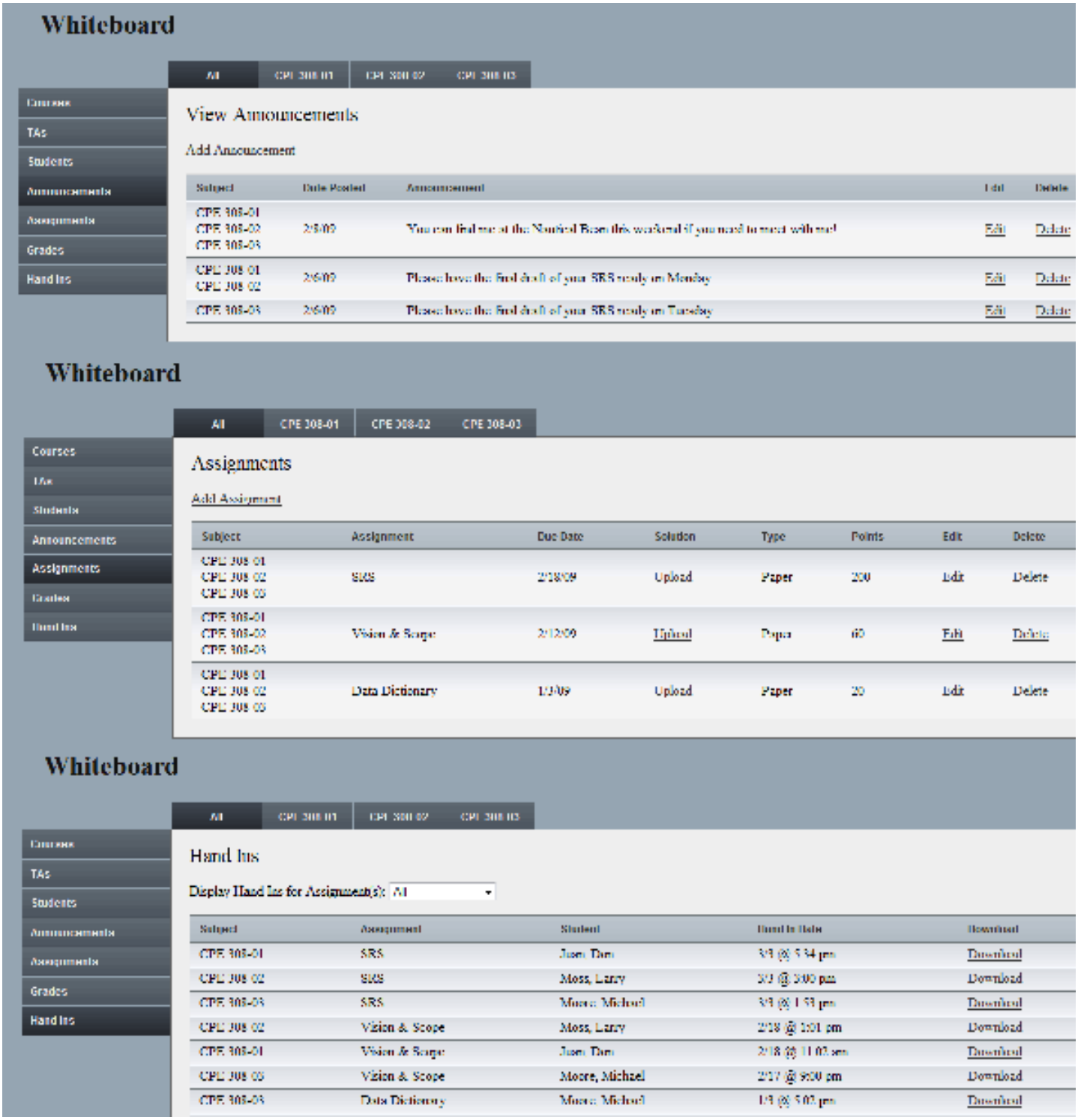

Course Info
Assignments List
View Assignments
Assignment Details
Student
Assignment Info
Assignment File
Grades
View Grades
Grades
View Metrics
Add Handin to Assignment
View Solutions
Solution File
(Uploaded Files)
Submit Handin
Submitted File

Appendix C: User Interface Prototypes

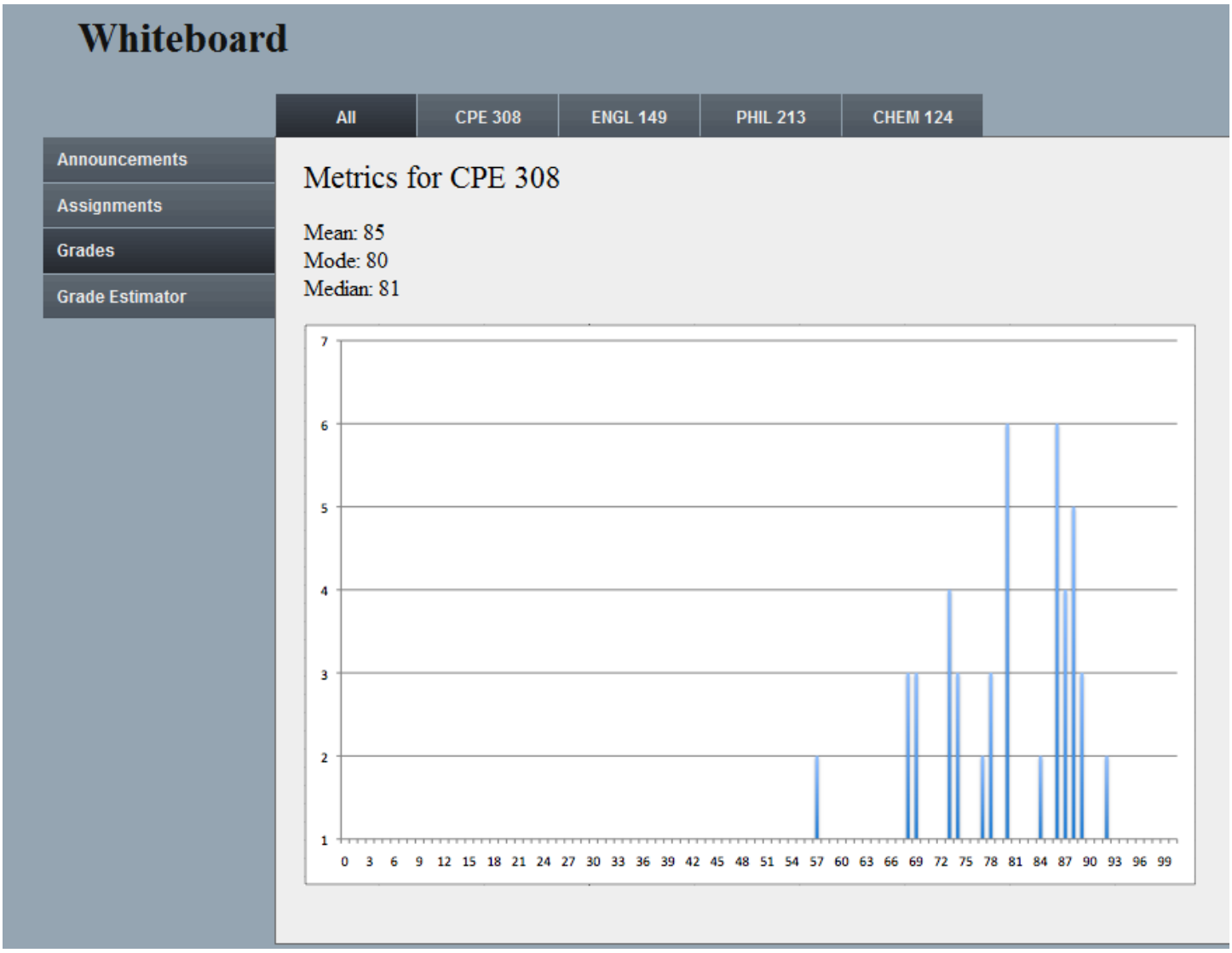
1. Student View



2. Professor View



3. *Metrics* Prototype



Appendix D: Market Research

Results Summary: General Consensus about Blackboard

Students	Professors
Horrible Layout	Cannot move layout around
Grades buried in the menus	Manually enter weights for each **assignment**
Good communication tools	Need **grade** feedback tool
Professors don't know how to use it	Too cumbersome, doesn't allow flexibility
Have to login into MyCalPoly, then log out if someone else used your computer and re-launch link.	Cannot standardize **grade** scheme for multiple sections

Professor Franz Kurfess

California Polytechnic State University
Computer Science Department
1 Grand Avenue
Building 14-218
San Luis Obispo CA 93407 U.S.A.
(805) 756-7179
fkurfess@csc.calpoly.edu

1.) What grading software do you currently use and why?
I primarily uses an excel spreadsheet and then transfers the **grade**s to Blackboard. My reasoning for using Excel is that it can perform virtually any calculation, especially since you can input custom formulas. I also use Excel because you can link to other spreadsheets easily.

2.) What features do you like about Blackboard?
I use Blackboard still because of its basic functionality; that is, it's useful for **student**s to check their **grade**s and **course**-related information. Another feature I like is the ability to create a repository for **assignment**s.

3.) What do you dislike about Blackboard?
It isn't feasible to state all of my dislikes about Blackboard, but the major problem is that BB is so cumbersome to use. I also dislike the how much navigating is necessary just for basic functionality (e.g. having to login to *Cal Poly* Portal first). Blackboard is missing an overview of new tasks/items.

4.) Do you have any suggestions for features in our system?
My classes are team based, like CPE 308, and so I would like a peer review tool (for the **student**s to use) which allows you to leave anonymous feedback about other teams or team members. I also suggest a possible groups feature – Blackboard has one but the term "nightmare" describes it.

Professor David B. Braun

Chair, Tenured & Probationary Faculty
Electrical Engineering Department
***Cal Poly* State University**
1 Grand Ave
Building 20-304
(805) 756-1464
dbraun@calpoly.edu

1.) What grading software do you currently use and why?
I use Excel to manage **student** scores on **assignment**s, exams, and projects and to calculate scores and help me assign **grade**s.

2.) What features do you like about Blackboard?
I use Blackboard sometimes to communicate pre-lab quiz scores to **student**s and sometimes to administer surveys and quizzes.

3.) What do you dislike about Blackboard?
Cal Poly's system requires me to enter each **grade** manually, which is a good check at the last step in assigning **grade**s.

4.) Do you have any suggestions for features in our system?
Excel is the most convenient. I can easily transfer scores to Excel from Blackboard and vice versa.

Paul R. Marchbanks, PhD
Assistant Professor of English
California Polytechnic State University
1 Grand Avenue 47-35A
San Luis Obispo, CA 93407-0322
805-756-2159
pmarchba@calpoly.edu

1.) What grading software do you currently use and why?
Blackboard

2.) What features do you like about Blackboard?
Ease of use and accessibility for **student**s. "Ease of use" means everything is intuitive for me at this point. I have no problem navigating Blackboard these days, given my frequency of use.

3.) What do you dislike about Blackboard?
I would like to be able to remove Blackboard's automatic averaging function, and rely on my own such column.

4.) Do you have any suggestions for features in our system?
I'd like not to have to click "okay" after every change I make.

Dr. Linda Bomstad, Professor Emeritus

Department of Philosophy
California Polytechnic State University, SLO
San Luis Obispo, CA 93407
805-756-7043
lbomstad@calpoly.edu

1.) What grading software do you currently use and why?
Blackboard; convenience

2.) What features do you like about Blackboard?
I like the way I can tailor it to my specific **assignment**s and their weighting.

3.) What do you dislike about Blackboard?
The one thing, the only thing really, that I dislike is that when I teach multiple sections of the same **course** (same **assignment**s, same weighting, etc.)I have to reconstruct my **grade** sheet each time.

4.) Do you have any suggestions for features in our system?
I wish there were some way to copy a **grade**book page, and then paste in into additional sections.

Professor John Oliver

Electrical Engineering Department
California Polytechnic State University
1 Grand Ave Room 314, Engineering East
San Luis Obispo, California 93407
(805) 756-5434
jyoliver@calpoly.edu

1.) What grading software do you currently use and why?
Yes Excel, for graphing anduploading to the registrar.

2.) What features do you like about Blackboard?
Ease of access for **student**s

3.) What do you dislike about Blackboard?
Not easy to manage multiple sections (post HW for two sections simultaneously).

4.) Do you have any suggestions for features in our system?
Can't place things on the blackboard web page where I want to. Entering **grade**s seems harder than in excel...hard to match up the names w/ the scores.

Professor Seiji A. Takemae

Physics Department
California Polytechnic State University
1 Grand Ave
San Luis Obispo, California 93407
stakemae@calpoly.edu

1.) What grading software do you currently use and why?
I use excel. I use it because it has functions (like average, standard deviation, or I can make my own functions to calculate the final **grade**) which can be applied to certain cells (of my choosing) in an array.

2.) What features do you like about Blackboard?
One major convenience is that on blackboard, I can download the class roster either as a pdf or an excel file. So, if I use Excel, I don't need to type in my class's roster. Furthermore, it speeds up my grading enormously. Let's suppose I have a roster of **student**'s names assorted by row and **grade**s assorted by column. In the last column, by typing in a single function (e.g. to calculate a **student**'s overall **coursegrade**) into one cell, I can then copy and paste that function into all the other cells in that same column to immediately get the overall **grade**s for all the other **student**s. This to me is probably the most important reason why I use Excel.

3.) What do you dislike (or would like to see improve) about it?
Right now, the faculty is using a PeopleSoft platform to access class rosters which includes the fields for **grade**s. Upon pushing an "approval" button, the data on the platform get sent to the registrar (at least as far as I understand). So far, I have had to input the **grade**s manually. It would be nice if I could just copy my excel spreadsheet which has my **student**s' names and **grade**s and paste it onto the file which goes to the registrar.

Professor Michael L. Haungs

Computer Science Department
California Polytechnic State University

1 Grand Ave Room 226, Frank E. Pilling
San Luis Obispo, California 93407
(805) 756-5531
mhaungs@calpoly.edu

1.) What grading software do you currently use and why?
I use Blackboard. It has all the features I need and **student**s can easily access their **grade**s. Plus, I don't have to concern myself with issues of posting **student** information publicly. Why reinvent the wheel?

2.) What features do you like about Blackboard?
I don't think any one feature is spectacular in Blackboard. I do, however, find great value in everything being integrated into one tool. I can display my syllabus and **assignment**s, provide class forums, and manage my **grade**s on one common platform that **student**s, by necessity, are all ready familiar with.

3.) What do you dislike about Blackboard?
The user interface is lackluster and setting up blackboard, from a new instructor's point-of-view, is not intuitive.

4.) Do you have any suggestions for features in our system?
I would like the ability to create a class "Template" and use it to instantiate new classes. I think the **course** copy mechanism of Blackboard is extremely clunky.

I would like an easy way to incorporate extra credit. Not only at the beginning of the **course**, but a nice, simple way to incorporate it if I decide to add it in, say, the 9th week.

Student Survey Question: What are you thoughts about Blackboard?

Student 1 (Major: Liberal Studies, Female):

- Doesn't like the wiki feature
- "I wish there was a button for **grade**s on the side instead of having to go through **Student** Tools"
- There's too much variability between how it's set up between different **course**s, meaning some **professor**s activate almost all the features, others just use one or two, while some don't use Blackboard altogether.

Student 2 (Major: Computer Science, Male)

- "It has a horrible layout. You have to navigate through too many pages to get where you want to go. For example, teachers barely use the **Announcement**s, yet it's the default home page."
- "It almost has too many features. Only the features that the teacher is using should be displayed. Otherwise, there ends up being too much clutter on the screen."
- "I like the message boards. Those can be very useful, especially for group projects."

Student 3 (Major: Forestry, Male)

- "There's a lot of stuff on there that's kind of redundant or not useful. For example, **Announcement**s, the **Student** Tools link is the same as the **Course** Tools link right below it, etc."
- "The Communication link is pretty cool and useful."
- "The pictures next to certain links are helpful, such as when you go to the Communication page."
- "Overall, the site could group things together better."

Student 4 (Major: Business, Female)

- Teachers don't know how to use it very well. Some put up **announcement**s everyday and all the **course** materials we will need for the quarter.
- It looks ugly. Not very professional. It only gets the job done.
- Things could be easier to access.

Student 5 (Major: Computer Engineering, Male)

- Finding your **grade**s is not obvious to first-time users.
- Having a separate Blackboard for every teacher is annoying.
- Some useful features require multiple clicks to get to.

Student 6 (Major: Mechanical Engineering, Male)

- There should be a **grade**s tab on the side for every class.
- The equation editor (used on the quizzes) is horrible, it has common symbols missing.

Appendix E: Comparison Matrix

	Competitors				Our Product
Company	**Pearson Education, Inc.**	**ThinkWave Inc.**		**Blackboard**	**Crosshair Solutions**
Web Site	www.pearsonschoolsystems.com	www.thinkwave.com		www.blackboard.com	http://wiki.csc.calpoly.edu/crosshairsolutions
Product Name	PowerSchool Premier	ThinkWave Educator		Academic Suite 6	Whiteboard 1.0
		Basic	Full		
Target Market					
Scope	Whole District	Single Teacher	District (fee for separate software)	Per-Teacher	Per-Teacher (all on same server)
School Type	K-12	K-12	K-12	K-12 and University	University
Interface	Web-based	Mixed		Web-based	Web-based
Server Platform	Windows Server with Java	Web is only for grade hosting		Windows, Linux, or Solaris; needs Java	Linux Java Application Server
Client Platform	Java Client	Teachers: Windows (web-based costs extra) Students: Web App		Web App	Web App.
Where Hosted	Own Server	Hosted by ThinkWave		Own Server	Own Server
Features: Available to Teachers					
Grade Views					
One-Student View	Yes	Yes		Yes	Yes
Class Averages	Yes	Yes		Yes	Yes
Whole-Class Table	Yes	Yes		Yes	Yes
Tools					
Upload assignment solutions	Unspecified	No		Yes	Yes
Send comments to students	Yes	Yes		Yes	Yes
Features: Available to Students					
Grade Views	Yes	*Ad-Supported*	Yes	Yes	Yes
Own Grades	Yes	Yes		Yes	Yes
Class Averages	Unspecified	No		Yes	Yes
All Students (without ID)	Unspecified	No		No	Yes
Tools					
View assignment solutions	Unspecified	No		Yes (different page)	Yes
Send comments to teachers	Yes	Yes		Yes	Yes
Features: Data-related					
Attendance	Yes	No	Yes	Unspecified	No
Printed reports	Yes	Yes		Yes	Yes
E-mail Reports	Yes	No	Yes	Unspecified	Yes
Generate Transcripts	Yes	Unspecified		Unspecified	No
Grading System					
Weighting and Curves					
Per assignment	Unspecified	Yes		*Choose Only One*	Yes
By Assn. Class / Type	Unspecified	Unspecified		*Choose Only One*	Yes
Overall-grade Curve	Unspecified	Yes		Unspecified	Yes
Statistics (Mean, Median, Mode)	Yes	Yes		Yes	Yes
Credit / No-Credit Assigmnents	Yes	Unspecified		Yes	Yes
Authentication					
Username + Password	Yes	Yes		Yes	Yes
Directory (LDAP) Server Integratio	Yes	Unspecified		No	No

Appendix F: Vision and Scope

1. Business Requirements

1.1 Background

According to our market survey, **student**s are often required to attend **professor**'s office hours to get their **grade**s (See Appendix D: Market Research). This puts an added burden on both the **professor** and the **student**, as the **student** has tomeet the **professor** at his office hours and the **professor** has to accommodate the **student**. If the **student** cannot make the office hours, the **student** and the **professor** must agree on another time.This requires extra time of the already busy schedules of **student**s and **professor**s. The **professor** would rather spend office hours on actual **course** content, not figuring out **grade**s. To eliminate this hassle, our client, WisdomWare Systems, has hired us, Crosshair **Solution**s, to aid with this dilemma.

An increasing number of **professor**s depend on software to keep track of their **student**s, **assignment**s, and **grade**s. According to our market research, most **professor**s use Excel or Blackboard to calculate **grade**s. The Excel file cannot be shared with the **student** due to FERPAlaws. If a **professor** modifies the file to meet FERPA regulations, then it can be posted on the web. But this requires the **professor** to update two files rather than one. Our market research shows most **professor**s dislike Blackboard because they have to enter the same **assignment** for multiple sections of the same **course**. Other software packages offer different features while sacrificing some (See Appendix E: Comparison Matrix). By having the software keep track of **course** administration, **professor**s can focus more on teaching in class.

Our client, WisdomWare Systems, Inc., has requested a custom software system which will minimize time spent on calculating **grade**s for **professor**s and **student**s. **Professor**s will save time and by using System to manage **course** enrollment, **assignment**, and **grade**. This will lead to **professor**s being able to use office hours to discuss **course** content, and more **course** time to teach rather than track **course** administration. The ability to access this information on the Internet or over a network would benefit both **professor**s and **student**s by reducing the time spent looking up **grade**s during office hours and by offering other important features to help **student**s with feedback.

1.2 Business Opportunity

The client is interested in a custom software **solution** because most existing software packages do not sufficiently meet the user's needs (See Appendix D: Market Research). For example, Blackboard has an average of four clicks per feature. According to our market survey, most users complain that they must log into the MyCalPoly Portal first. This increases the time the user has to wait for their **grade**s. In addition, most are targeted at university administrators rather than **professor**s.Our market survey shows that **professor**s dislike having to re-input **assignment**s and **solution**s for multiple sections of the same **course**. The client desires a system that addresses these problems and provides a graphical user interface that minimizes the amount of time and number of clicks required to execute each use.

1.3 Business Objectives and Success Criteria

BO-1: Reduce the number of hours spent by **student** attending **professor**'s office hours by 10% per quarter by the end of the first year.

BO-2: Increase the number of hours spent by **professor** discussing **course** content during office hours by 10% per quarter by the end of first year.

BO-3: Reduce the number of **assignment**s turned in late by 10% each quarter by the end of the first year.

BO-4: Reduce the amount of time spent by **professor**s recording and computing **grade**s by 10% per quarter by the end of the first year.

BO-5: Increase amount of time spent by **professor**s teaching during **course** by 10% per quarter by the end of the first year.

SC-1: Have 10 or more **professor**s using System by the end of the first year.

SC-2: All features can be accessed in three clicks or less.

1.4 Customer or Market Needs

CN-1: Trials demonstrate that the user interface facilitates the execution of all use cases in a less than five minutes and three clicks or less.

CN-2: **Professor**s can manage **course** enrollment, **assignment**, and grading data.

CN-3: **Student**s can access **assignment**s, **grade**s, and view *metrics*, outside of **professor**'s office hours.

CN-4: **Professor**s can communicate all **student**s enrolled in a **course** via an integrated **announcement** system.

1.5 Business Risks

RI-1: Development may not be completed by the deadline, resulting in a later release date and higher costs.

RI-2: **Professor**s may choose not to use this system because they feel it is not secure. **Grade**s might be kept both electronically and in a **grade** book, which would inevitably lead to the electronic system becoming an inconvenience. To allay this risk we will follow a well-known security standard and educate users about the product's security features through marketing and a built-in help or FAQ.

RI-3: Competitors may offer a similar product at a lower price. (The price of the product will be determined by the client.) Increasing advertising or lowering the price of the product are possible **solution**s.

RI-4: **Student**s may choose not to use System due to unfamiliarity. Instead of inquiring about their **grade**, they might ask the **professor** how to use the grading tool, creating more frustration for both **professor**s and **student**s. We will address this risk by building a help and FAQ system into the product, and by performing research to refine the user interface based on feedback from **professor**s and **student**s.

RI-5: A larger company may gain knowledge of our system design and buyout our contract with the customer.

2. Vision of the Solution

2.1 Vision Statement

We will build an online system that **professor**s can use to post **grade**s and **student**s can use to view their **grade**s. In addition, the system will generate statistics and visualizations (such as graphs or charts) of **grade** data for both **student**s and **professor**s.

2.2 Major Features

FE-1: Create and modify **course**s

FE-2: Create and modify enrollment for each **course**

FE-3: Enter and modify **assignment** information, and attach accompanying files

FE-4: Provide visualization of the distribution of **grade**s (e.g. a pie chart)

FE-5: Post **announcement**s for a **course**, visible to all **student**s enrolled in that **course**

FE-6: Enter and manage **grade**s

FE-7: Set curve for **course**

FE-8: **Grade** Estimator

FE-9: **Hand-In**

FE-10: Manage **TA**s

2.3 Assumptions and Dependencies

AS-1: A **student** roster will be provided to store in our System

DE-1: All **student**s will have a computer readily available with internet access

DE-2: Access to the accurate **student** and **course** data from the SIS.

3. Scope and Limitations

3.1 Scope of Initial and Subsequent Releases

Feature	Release 0.1 (Alpha)	Release 0.2 (Beta)	Release 1.0 (Final)
FE-1	*Upload roster from database into grading tool; Drop **student**s from **course** option (but not from school registrar)*	*Drop **student**s through the Cal PolyRegistrar*	*Verify enrolled **student**s with Cal Poly Registrar (Implemented if time permits)*
FE-2	*Fully Implemented*		
FE-3	*Fully Implemented*		
FE-4	***Assignment**/Exam statistics, e.g. average **grade**, displayed in tabular format*	*Statistics displayed in charts and graphs*	*Fully Implemented*
FE-5	*Implemented if time permits (medium priority)*	*Implemented if time permits (low priority)*	*Implemented if time permits (low priority)*
FE-6	*Fully Implemented*		
FE-7	*Implemented if time permits (medium priority)*	*Fully Implemented*	
FE-8		*Implemented if time permits (medium priority)*	*Fully Implemented*
FE-9		*Implemented if time permits (medium priority)*	*Fully Implemented*

3.2 Limitations and Exclusions

LI-1: Whiteboard™ shall only be guaranteed compliant with the California PolytechnicState University, San Luis Obispo electronic **student** information system (SIS).

LI-2: The language of the product and all of its features will be exclusively American English.

LI-3: System will not enforce **course** prerequisites.

EX-1: **Student** personal and academic information will not be displayed to any user other than him, and those permitted in accordance with FERPA.

EX-2: Whiteboard™ willbe unusable without an Internet connection. Local copy of data will not be kept.

4. Business Context

4.1 Stakeholder Profiles

Stakeholder	*Major Value*	*Attitudes*	*Major Interests*	*Constraints*
WisdomWare Systems, Inc.	*Use of grading tool software product for schools to be sold for profit*	*Strong commitment through release 1.0*	*Cost savings must exceed development and usage costs*	*None identified*
Project Manager (Dr. Turner)	*Development team experiences software development process and learns from it*	*Strong commitment through all releases*	*Satisfaction of the customer; successful development team collaboration and planning*	*Unable to oversee implementation and later phases*
***Professor**s*	*More organized **grade** book; simplified number of tools (grading, add/dropping **student**s, communication, attendance); increased efficiency*	*Some concerned about own lack of technical knowledge; some fail to see advantages over standard medium (paper); others receptive*	*Reliability; reduced workload compared to competitors' products; reduce admin, increase **course** content time*	*Training for staff of grading tool functionality*
TAs (Teacher Assistants)	*Convenience; ability to use most of the same valuable resources available to **professor**s*	*Strong enthusiasm, but concerned about low usage by the **professor** whom they work for*	*Reduced workload compared to competitors' products*	***Professor** must grant access*
***Student**s*	*Access to **assignment** statistics; ability to monitor **course** progress*	*Some potentially uninterested in learning to use a new grading tool; most other **student**s receptive*	*More inclusive and informational grading system to monitor progress within **course**s*	***Professor** must use software and post **grade**s consistently*

4.2 Project Priorities

Dimension	Driver	Constraint	Degree of Freedom
Schedule	*SRS completed by 2/13/09; grading tool release 1.0 completed before 6/05/09*	*Final version (release 1.0) available by 6/05/09*	*Release 0.1 (Alpha) planned to be available between 4/13/09 and 5/11/09; Release 0.2 (Beta)to be available between 5/11/09 and 6/05/09*
Features	*All features scheduled for release 1.0 fully operational*	*All features scheduled for release 0.1 must be fully operational*	
Quality	*Grading tool responds to user's request within the agreed upon timeframe (4 seconds proposed)*	*Available hardware may prevent desired response time; grading tool must respond to user's request within 8 seconds (proposed) to be considered practical*	
Staff	*Projected team size is six, with one taking the role of project manager*	*Potential change in staff midway through project (transition to CPE 309)*	
Cost (Time)	*Staff should preserve at least 10 hrs for work outside of the classroom per week*	*Staff cannot fully dedicate themselves to the project due to other external priorities (**course** load)*	

4.3 Operating Environment

The grading tool will be a web-based application, requiring a connection to the Internet or the school network in order to retrieve stored information from the database. Users will have the ability to remotely access the grading tool through use of a web browser (Internet Explorer 7.0 and Mozilla Firefox 3.0 guaranteed). The grading tool shall be available all hours of the day, unless the product is offline as a result of maintenance. Every **student**, **professor**, and **TA** will have a unique log-in name and password that will be used in order to access the product. Information sent to/from and stored by System will be encrypted for the security of the user.

Appendix G: Nonfunctional Requirements Notes

1. Compatible Format

System shall be capable of parsing tab-separated files (.tsv). The following is an example of a parsable format (tab characters separate each item):

Term	Crs Id	Section Id	Activity Type	Instructor
20044	CPE 0205	01	LEC	Fisher G

Student Name	**Student** Id	Major	Email Address
"Avery, Jonathan Robert"	xxxxxxxxx	CSC	jravery@calpoly.edu
"Baradaran, Shahram"	xxxxxxxxx	CSC	sbaradar@calpoly.edu
"Baughman, Kendra Rose"	xxxxxxxxx	CSC	kbaughma@calpoly.edu
"Campbell, Brian S."	xxxxxxxxx	CSC	bscampbe@calpoly.edu
"Cronquist, Mark Herbert"	xxxxxxxxx	CSC	mcronqui@calpoly.edu
"Curtis, Tyler J."	xxxxxxxxx	CSC	tjcurtis@calpoly.edu
"Cushing, Kyle Robert"	xxxxxxxxx	CSC	cushing@calpoly.edu
"Daniel, Scott Andrew"	xxxxxxxxx	CSC	sadaniel@calpoly.edu
"Davis, Blake Joseph"	xxxxxxxxx	CSC	bjdavis@calpoly.edu
"Dennahower, Justin Michael"	xxxxxxxxx	CSC	jdennaho@calpoly.edu
"Do, Manh Cuong Nguyen"	xxxxxxxxx	CSC	mdo@calpoly.edu
"Dukovich, Adam Matthew"	xxxxxxxxx	CSC	adukovic@calpoly.edu
"Griffin, Scott Patrick"	xxxxxxxxx	CSC	sgriffin@calpoly.edu
"Guest, Ryan James"	xxxxxxxxx	CSC	rguest@calpoly.edu
"Lawrence, Peter John"	xxxxxxxxx	CPE	plawrenc@calpoly.edu
"Maas, Craig Alan"	xxxxxxxxx	CSC	cmaas@calpoly.edu
"Macauley, James Arthur"	xxxxxxxxx	CSC	jamacaul@calpoly.edu
"McCabe, Keith Edward"	xxxxxxxxx	CSC	kmccabe@calpoly.edu
"Nichols, Ashley Anne"	xxxxxxxxx	CSC	anichols@calpoly.edu
"Nong, Hong Le"	xxxxxxxxx	CSC	hnong@calpoly.edu
"Ober, Timothy Robert"	xxxxxxxxx	CSC	tober@calpoly.edu
"Pawlik, Jennifer Michelle"	xxxxxxxxx	CSC	jpawlik@calpoly.edu
"Schenkel, Clay Andrew"	xxxxxxxxx	CSC	cschenke@calpoly.edu
"Sena, Joseph Andrew"	xxxxxxxxx	CSC	jasena@calpoly.edu
"St John, Douglas Allen"	xxxxxxxxx	CSC	dstjohn@calpoly.edu
"Tran, PhucDanh Le"	xxxxxxxxx	CSC	ptran@calpoly.edu
"Vaughan, Benjamin William"	xxxxxxxxx	CSC	bvaughan@calpoly.edu
"Yang, Chuang Sheng"	xxxxxxxxx	CSC	csyang@calpoly.edu

2. Justification of Design for a Tabbed Interface

Advantages:
Tabs convey information quickly to users due to their organized, intuitive style. Identical to physical filing systems, tabs isolate sections of information and provide a quick method of locating sought after records/documents.

Disadvantages:
Problems arise when the number of tabs associated with an interface becomes too great. If a window holds more tabs than can fit on the screen at once (meaning horizontal scrolling is necessary), then their effectiveness is reduced. Furthermore, multiple rows of tabs clutter the screen and prohibit quick communication to the user.

Alternative Design:
We considered the use of drop down menus in the interface since they present a clean, crisp design. However, we found they hide available options until they have been activated, forcing the user to search the display more frequently. In addition, selecting items on the drop down menu adds more time to tasks in comparison to the same functionality achieved by the single click of a tab.

Conclusion:
Since we anticipate a limited number of **course**s (five on average) taken by the **student** or instructed by the **professor**, we feel that a tabbed interface will provide the most effective means to communicate options to the user. User uncertainty will be low since all possible choices will always be displayed to **professor**s, **student**s, and **TA**s. Such a design will limit user frustration, and improve the product's likelihood of success.

Appendix H: Use Cases

Use Case ID:	1.0		
Use Case Name:	View **Course**s		
Created By:	Wilson Lau	Last Updated By:	Jason Boyle
Date Created:	January 28, 2009	Date Last Updated:	February 25, 2009
Actors:	**Professor**, **TA**		
Description:	Actor views information about **course**s.		
Preconditions:	1. Actor is logged into System.		
Postconditions:	None		
Normal Flow:	**1.0View Course**s 1. Actor requests to view **course**s. 2. System displays information about **course**s *relevant* to Actor.		
Alternative Flows:	**1.0.A.1 View a Single Course** (after step 2) 1. Actor requests to view a **course**. 2. System displays **course** data.		
Exceptions:	**1.0.E.1 No Courses Exist** (at step 2) 1. System informs Actor that no **course**s exist. 2. Use case is terminated.		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	1.1		
Use Case Name:	Add **Course**		
Created By:	Wilson Lau	Last Updated By:	Jason Boyle
Date Created:	January 28, 2009	Date Last Updated:	February 25, 2009
Actors:	**Professor**		
Description:	Actor adds a **course**.		
Preconditions:	1. Actor is logged into System.		
Postconditions:	1. **Course** is stored by System.		
Normal Flow:	**1.1 Add a Course** 1. Actor requests to add a **course**. 2. System displays a form with inputs fields for **course** data. 3. Actor enters **course** data. 4. Actor indicates that he has finished entering **course** data. 5. System stores the **course**.		
Alternative Flows:	**1.1.A.1 Cancel Add**(after step 2) 6. Actor requests to cancel adding the **course**. 7. Use case is terminated.		
Exceptions:	**1.1.E.1** *Invalid* **Course Data is Entered** (at step 4) 1. System notifies Actor as to which **course** data is *invalid*. 2. Return to step 3.		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	1.2		
Use Case Name:	Update **Course**		
Created By:	Wilson Lau	Last Updated By:	Jason Boyle
Date Created:	January 28, 2009	Date Last Updated:	February 25, 2009
Actors:	**Professor**		
Description:	Actor updates a **course**.		
Preconditions:	1. Actor is logged into System. 2. At least one **course** *relevant* to Actor exists.		
Postconditions:	1. **Course** is updated by System.		
Normal Flow:	**1.2 Update a Course** 1. Actor requests to update a **course**. 2. System displays a form with inputs for **course** data preset to existing data. 3. Actor alters **course** data. 4. Actor indicates that he has finished altering **course** data. 5. System updates the **course**.		
Alternative Flows:	**1.2.A.1 Cancel Update** (after step 2) 8. Actor requests to cancel updating the **course**. 9. Use case is terminated.		
Exceptions:	**1.2.E.1** *Invalid* **Course Data is Entered** (at step 4) 1. System notifies Actor as to which **course** data is *invalid*. 2. Return to step 3.		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	1.3		
Use Case Name:	Delete **Course**		
Created By:	Wilson Lau	Last Updated By:	Jason Boyle
Date Created:	January 28, 2009	Date Last Updated:	February 25, 2009
Actors:	**Professor**		
Description:	Actor deletes a **course**.		
Preconditions:	1. Actor is logged into System. 2. At least one **course** *relevant* to Actor exists.		
Postconditions:	1. **Course** is deleted by System.		
Normal Flow:	**1.3 Delete a Course** 1. Actor requests to delete a **course**. 2. System prompts Actor to confirm deletion. 3. Actor confirms that he wants to delete the **course**. 4. System deletes the **course**.		
Alternative Flows:	**1.3.A.1 Cancel Delete**(after step 1) 1. Actor requests to cancel deleting the **course**. 2. Use case is terminated.		
Exceptions:	None		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	1.4		
Use Case Name:	View **Course** *Metrics*		
Created By:	Wilson Lau	Last Updated By:	Jason Boyle
Date Created:	January 28, 2009	Date Last Updated:	February 25, 2009
Actors:	**Professor**		
Description:	System displays *metrics* for a **course**.		
Preconditions:	10. Actor is logged into System. 11. At least one **course** *relevant* to Actor exists. 12. At least one **student** is enrolled in at least one the *relevant* **course**s. 13.		
Postconditions:	None		
Normal Flow:	**1.4 View Course *Metrics*** 1. Actor requests to view *metrics* for a **course**. 2. System displays *metrics* for the **course**.		
Alternative Flows:	None		
Exceptions:	**1.4.A.1No Course Data Exists** (after step 2) 1. System informs Actor that no **assignment**s for the **course** have been **grade**d. 2. Use case is terminated.		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	2.0		
Use Case Name:	View **TA**s		
Created By:	Wilson Lau	Last Updated By:	Jason Boyle
Date Created:	January 29, 2009	Date Last Updated:	February 25, 2009
Actors:	**Professor**		
Description:	System displays TAs currently assisting Actor in one or more of the **course**s he is teaching.		
Preconditions:	1. Actor is logged into System. 2. At least one **TA** *relevant* to Actor exists.		
Postconditions:	None		
Normal Flow:	**2.0 View TA**s 1. Actor requests to view **TA**s. 2. System displays **TA**s *relevant* to the Actor and the **course** with which each **TA** is currently assisting.		
Alternative Flows:	None		
Exceptions:	**2.0.E.1 No TAs Exist**(at step 2) 1. System notifies Actor that he currently has no **TA**s. 2. Use case is terminated.		
Priority:	Medium		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		

Notes and Issues: None

Use Case ID:	2.1		
Use Case Name:	Add **TA**		
Created By:	Wilson Lau	Last Updated By:	Jason Boyle
Date Created:	January 29, 2009	Date Last Updated:	February 25, 2009
Actors:	**Professor**		
Description:	Actor adds a **TA**.		
Preconditions:	1. Actor is logged into System. 2. At least one **TA** *relevant* to Actor exists.		
Postconditions:	1. **TA** is stored by System.		
Normal Flow:	**2.1 Add TA to Course** 1. Actor requests to add a **TA**. 2. System displays a form with inputs fields for **TA** data. 3. Actor enters **TA** data. 4. Actor indicates that he is finished entering **TA** data. 5. System stores the **TA**.		
Alternative Flows:	**2.1.A.1 Cancel Add**(after step 1) 1. Actor requests to cancel adding the **TA**. 2. Use case is terminated.		
Exceptions:	**2.1.E.1 Maximum Number of TA**s **Exceeded** (at step 1) 1. System notifies Actor that another **TA** cannot be added without exceeding the maximum number of **TA**s for a **professor**. 2. Use case is terminated. **2.1.E.2***Invalid***TA Data is Entered** (at step 4) 1. System notifies Actor as to which **TA** data is *invalid*. 2. Return to step 3.		
Priority:	Medium		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	2.2		
Use Case Name:	Update **TA**		
Created By:	Jason Boyle	Last Updated By:	n/a
Date Created:	February 25, 2009	Date Last Updated:	n/a
Actors:	**Professor**		
Description:	Actor updates a **TA**.		
Preconditions:	1. Actor is logged into System. 2. At least one **TA** *relevant* to the Actor exists.		
Postconditions:	1. **TA** is updated by System.		
Normal Flow:	**2.2UpdateTA** 1. Actor requests to update a **TA**. 2. System displays a form with inputs for **TA** data preset with existing data. 3. Actor alters **TA** data. 4. Actor indicates that he is finished altering **TA** data. 5. System updates the **TA**.		
Alternative Flows:	**2.2.A.1 Cancel Update**(after step 1) 1. Actor requests to cancel updating the **TA**. 2. Use case is terminated.		
Exceptions:	**2.1.E.1***Invalid***TA Data is Entered** (at step 4) 1. System notifies Actor as to which **TA** data is *invalid*. 2. Return to step 3.		
Priority:	Medium		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	2.3		
Use Case Name:	Delete **TA**		
Created By:	Wilson Lau	Last Updated By:	Jason Boyle
Date Created:	January 29, 2009	Date Last Updated:	February 25, 2009
Actors:	**Professor**		
Description:	Actor deletes a **TA**.		
Preconditions:	1. Actor is logged into System. 2. At least one **TA** *relevant* to Actor exists.		
Postconditions:	1. **TA** is deleted by System.		
Normal Flow:	**2.3 Delete TA** 1. Actor requests to delete a **TA**. 2. System prompts Actor to confirm deletion. 3. Actor confirms that he wants to delete the **TA**. 4. System deletes the **TA**.		
Alternative Flows:	**2.3.A.1 Cancel Delete**(after step 1) 1. Actor requests to cancel deleting the **TA**. 2. Use case is terminated.		
Exceptions:	**2.3.E.1 No TA to Delete** (at step 1) 1. System notifies Actor that he currently has no **TA**s. 2. Use Case is terminated.		
Priority:	Medium		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	3.0		
Use Case Name:	View **Student**s		
Created By:	Wilson Lau	Last Updated By:	Jason Boyle
Date Created:	January 29, 2009	Date Last Updated:	February 25, 2009
Actors:	**Professor**		
Description:	Actor views information about the **student**s in a **course**.		
Preconditions:	1. Actor is logged into System. 2. At least one **student** is enrolled in at least one **course** *relevant* to Actor.		
Postconditions:	None		
Normal Flow:	**3.0 View Student**s 1. Actor requests to view **student**s for a **course**. 2. System displays information about **student**s enrolled in the **course**.		
Alternative Flows:	**3.0.A.1 View Single Student** (after step 1) 1. Actor requests to view a **student**. 2. System displays information about the **student**.		
Exceptions:	**3.0.E.1 No Students Exist**(at step 1) 1. System displays a message notifying Actor that no **student**s are enrolled in the **course**. 2. Use case is terminated.		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	3.1		
Use Case Name:	Add **Student**		
Created By:	Wilson Lau	Last Updated By:	Jason Boyle
Date Created:	January 29, 2009	Date Last Updated:	February 25, 2009
Actors:	**Professor**		
Description:	Actor adds a **student**.		
Preconditions:	1. Actor is logged into System. 2. At least one **course** *relevant* to Actor exists.		
Postconditions:	1. **Student** is stored by System.		
Normal Flow:	**3.1 Add Student to Course** 1. Actor requests to add a **student** to a **course**. 2. System displays form with inputs fields for **student** data. 3. Actor enters **student** data. 4. Actor indicates that he has finished entering **student** data. 5. System stores **student**.		
Alternative Flows:	**3.1.A.1 Cancel Add**(after step 1) 1. Actor requests to cancel adding the **student**. 2. Use case is terminated.		
Exceptions:	**3.1.E.1** *Invalid* **Course Data is Entered** (at step 4) 1. System notifies user as to which **student** data is *invalid*. 2. Return to step 3. **3.1.E.2 Maximum Number of Students Exceeded**(at step 1) 1. System notifies Actor that another **student** cannot be added without exceeding the maximum number of **student**s allowed for a **course**. 2. Use case is terminated.		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	3.2		
Use Case Name:	Update **Student**		
Created By:	Wilson Lau	Last Updated By:	Wilson Lau
Date Created:	January 26, 2009	Date Last Updated:	February 26, 2009
Actors:	**Professor**		
Description:	Actor updates a **student**.		
Preconditions:	1. Actor is logged into System. 2. At least one **student** *relevant* to Actor exists.		
Postconditions:	1. **Student** is updated by System.		
Normal Flow:	**3.2 Update a Course** 1. Actor requests to update a **student**. 2. System displays a form with inputs for **student** data preset to existing data. 3. Actor alters **student** data. 4. Actor indicates that he has finished altering **student** data. 5. System updates the **student**.		
Alternative Flows:	**3.2.A.1 Cancel Update** (after step 2) 1. Actor requests to cancel updating the **student**. 2. Use case is terminated.		
Exceptions:	**3.2.E.1** *Invalid* **Course Data is Entered** (at step 4) 1. System notifies Actor as to which **student** data is *invalid*. 2. Return to step 3.		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	3.3		
Use Case Name:	Delete **Student**		
Created By:	Wilson Lau	Last Updated By:	Jason Boyle
Date Created:	January 29, 2009	Date Last Updated:	February 26, 2009
Actors:	**Professor**		
Description:	Actor deletes a **student** from System.		
Preconditions:	1. Actor is logged into System. 2. At least one **student** *relevant* to Actor exists.		
Postconditions:	1. **Student** is deleted by System.		
Normal Flow:	**3.3 Delete Student from Course** 1. Actor requests to delete a **student**. 2. System prompts Actor to confirm deletion. 3. Actor confirms that he wants to delete the **student**. 4. System deletes the **student**.		
Alternative Flows:	None		
Exceptions:	**3.3.A.1 Cancel Delete**(after step 1) 1. Actor requests to cancel deleting the **student**. 2. Use case is terminated.		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	3.4		
Use Case Name:	Import Roster		
Created By:	Wilson Lau	Last Updated By:	Jason Boyle
Date Created:	January 29, 2009	Date Last Updated:	February 26, 2009
Actors:	**Professor**		
Description:	Actor adds **student**s to a **course** by importing data from an external file.		
Preconditions:	1. Actor is logged into System. 2. At least one **course** *relevant* to Actor exists in System.		
Postconditions:	1. **Student**s are stored by System.		
Normal Flow:	**3.4 Import Student Roster** 1. Actor requests to import a **student** roster for a **course**. 2. System prompts actor to select a file containing data for **student**s. 3. Actor selects a file. 4. System imports file data and stores the corresponding **student**s.		
Alternative Flows:	**3.4.A.1Student**s **Already Enrolled in Course**(at step 1) 1. System warns Actor that all **student**s currently enrolled in the **course** will be deleted along with all corresponding **grade** data. 2. Return to step 2.		
Exceptions:	**3.4.E.1** *Invalid* **File Format** (at step 4) 1. System notifies Actor that the format of the selected file is not supported, and displays the selected file type and the supported file types for comparison. 2. Return to step 2. **3.4.E.2Corrupted File**(at step 4) 1. System notifies Actor that the selected file is corrupt or incorrectly formatted. 2. Return to step 2. **3.3.A.1 Cancel Import**(after step 1 and step 1 of 3.4.A.1) 1. Actor requests to cancel importing roster. 2. Use case is terminated.		
Priority:	High		
Business Rules:	None		
Special Requirements:	If a **student** exists in the **course** and in a new roster, he will remain in the **course**. If a **student** exists in the **course**, but not in a new roster, he will be removed from the **course**. If a **student** does not exist in the **course**, but is in a new roster, he will be added to the **course**.		
Assumptions:	DE-2		
Notes and Issues:	Actor will be clearly warned that existing **student**s in the **course** will be deleted and given the option to cancel the import at any time.		

Use Case ID:	3.5		
Use Case Name:	View **Student** *Metrics*		
Created By:	Wilson Lau	Last Updated By:	Jason Boyle
Date Created:	January 29, 2009	Date Last Updated:	February 26, 2009
Actors:	**Professor**		
Description:	System displays *metrics* for a **student**.		
Preconditions:	1. Actor is logged into System. 2. A least one **student** *relevant* to Actor exists.		
Postconditions:	None		
Normal Flow:	**3.5 View Course *Metrics*** 1. Actor requests to view *metrics* for a **student**. 2. System displays *metrics* for the **student**.		
Alternative Flows:	None		
Exceptions:	None		
Priority:	Medium		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	4.0		
Use Case Name:	View **Announcement**s		
Created By:	Jason Boyle	Last Updated By:	Jason Boyle
Date Created:	February 9, 2009	Date Last Updated:	February 26, 2009
Actors	**Professor**, **TA**, **Student**		
Description:	Actor views **announcement**s posted by the **Professor** or **TA** for one or more **course**s.		
Preconditions:	1. Actor is logged into System.		
Postconditions:	None		
Normal Flow:	**4.0 View Announcements** 1. Actor requests to view **announcement**s. 2. System displays **announcement**s for **course**s *relevant* to Actor in chronological order.		
Alternative Flows:	**4.0.A.1 View Announcements for a Single Course** (after step 2) 1. Actor requests to view **announcement**s for a single **course**. 2. System displays **announcement**s for the selected **course** in chronological order.		
Exceptions:	**4.0.E.1 No Announcements Exist** (at step 2) 1. System informs Actor that there are no **announcement**s. 2. Use case is terminated.		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	4.1		
Use Case Name:	Add **Announcement**		
Created By:	Nestor Reyes	Last Updated By:	Jason Boyle
Date Created:	February 7, 2009	Date Last Updated:	February 26, 2009
Actors:	**Professor**		
Description:	Actor adds an **announcement** to a **course**.		
Preconditions:	1. Actor is logged into System.		
Post conditions:	1. **Announcement** is stored by System. 2. **Announcement** is e-mailed to every **student** enrolled in the **course**.		
Normal Flow:	**4.1 Create Announcement** 1. Actor requests to add an **announcement** for a **course**. 2. System displays a form with inputs fields for **announcement** data. 3. Actor enters **announcement** data. 4. Actor indicates that he has finished entering data. 5. System stores the **announcement**. 6. System e-mails **announcement** to every **student** enrolled in the **course**.		
Alternative Flows:	**4.1.A.1 Cancel Add**(after step 1) 1. Actor requests to cancel adding the **announcement**. 2. Use case is terminated.		
Exceptions:	**4.1.E.1 Announcement Text Exceeds Maximum (1024) Character Limit** 1. System notifies Actor that the **announcement** exceeds the maximum character limit. 2. Return to step 3.		
Includes:	None		
Priority:	High		
Business Rules:	None		
Special Requirements:	1. Actor shall be able to prevent the System from e-mailing the **announcement** to **student**s.		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	4.2		
Use Case Name:	Update **Announcement**		
Created By:	Nestor Reyes	Last Updated By:	Jason Boyle
Date Created:	February 7, 2009	Date Last Updated:	February 26, 2009
Actors:	**Professor**		
Description:	Actor edits an **announcement**.		
Preconditions:	1. Actor is logged into System. 2. At least one **announcement** *relevant* to Actor exists.		
Post conditions:	1. **Announcement** is updated by System.		
Normal Flow:	**4.2 Update Announcement** *1.* Actor requests to update an **announcement**. *2.* System displays a form with inputs fields for **announcement** data preset to existing data. *3.* Actor enters **announcement** data. *4.* Actor indicates that he has finished entering **announcement** data. *5.* System stores the **announcement**.		
Alternative Flows:	**4.2.A.1 Cancel Update**(after step 1) 1. Actor requests to cancel updating the **announcement**. 2. Use case is terminated.		
Exceptions:	**4.2.E.2 Announcement Text Exceeds Maximum (1024) Character Limit** 1. System notifies Actor that the **announcement** exceeds the maximum character limit. 2. Return to step 3.		
Priority:	Medium		
Business Rules:	None		
Special Requirements:	1. Actor shall have the option of e-mailing the updated **announcement** to all **student**s.		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	4.3		
Use Case Name:	Delete **Announcement**		
Created By:	Nestor Reyes	Last Updated By:	Jason Boyle
Date Created:	February 7, 2009	Date Last Updated:	February 26, 2009
Actors:	**Professor**		
Description:	Actor deletes an **announcement**.		
Preconditions:	1. Actor is logged into System. 2. At least one **announcement** *relevant* to Actor exists.		
Post conditions:	1. **Announcement** is deleted by System.		
Normal Flow:	**4.3 Delete Announcement** 1. Actor requests to delete an **announcement**. 2. System prompts Actor to confirm deletion. 3. Actor confirms that he wants to delete the **announcement**. 4. System deletes the **announcement**.		
Alternative Flows:	**4.3.A.1 Cancel Update**(after step 1) 1. Actor requests to cancel deleting the **announcement**. 2. Use case is terminated.		
Exceptions:	None		
Priority:	Medium		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	5.0		
Use Case Name:	View **Assignment**s		
Created By:	Wilson Lau	Last Updated By:	Jason Boyle
Date Created:	January 28, 2009	Date Last Updated:	February 26, 2009
Actors:	**Professor**, **TA**, **Student**		
Description:	Actor views **assignment**s for all **course**s or a single **course**.		
Preconditions:	1. Actor is logged into System.		
Postconditions:	None		
Normal Flow:	**5.0 View Assignments** 1. Actor requests to view **assignment**s. 2. System displays information about **assignment**s for **course**s*relevant* to Actor in chronological order.		
Alternative Flows:	**5.0.A.1 View Assignments for a Course** (after step 1) 1. Actor requests to view **assignment**s for a **course**. 2. System displays **assignment**s data for the **course** in chronological order. **5.0.A.2 View Assignments for a Course** (after step 1 or step 1 of 5.0.A.1) 1. Actor requests to view an **assignment**. 2. System displays **assignment** data.		
Exceptions:	None		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	5.1		
Use Case Name:	Add **Assignment**		
Created By:	Wilson Lau	Last Updated By:	Wilson Lau
Date Created:	January 28, 2009	Date Last Updated:	February 09, 2009
Actors:	**Professor**, System		
Description:	Actor adds an **Assignment**s to System.		
Preconditions:	1. Actor is logged into System.		
Postconditions:	1. **Assignment** becomes added to System.		
Normal Flow:	**5.1.0 Add a Assignment** 1. Actor requests to add an **Assignment**. 2. Actor enters in **Assignment**. 3. Actor confirms to system that he wants to add **Assignment**. 4. System stores **Assignment**.		
Alternative Flows:	None		
Exceptions:	**5.1.0.E.1** *Invalid* **Assignment is Entered** (at step 2) 1. System notifies Actor that *invalid* **assignment** has been entered by displaying error message "**Assignment** data *invalid*." 2. Actor changes **assignment** data in question. 3. Return to step 2.		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	5.2		
Use Case Name:	Update **Assignment**		
Created By:	Wilson Lau	Last Updated By:	Wilson Lau
Date Created:	January 28, 2009	Date Last Updated:	February 09, 2009
Actors:	**Professor**		
Description:	Actor updates an **assignment**s to System.		
Preconditions:	1. Actor is logged into System.		
Postconditions:	1. **Assignment** becomes updated in System.		
Normal Flow:	**5.2.0 Update a Assignment** 1. Actor requests to updates an **assignment**. 2. Actor chooses an **assignment** to update. 3. Actor updates the **assignment**. 4. Actor confirms to system that he wants to update **assignment**. 5. System updates **assignment**.		
Alternative Flows:			
Exceptions:	**5.2.0.E.1** *Invalid* **Assignment is Entered** (at step 3) 1. System notifies Actor that *invalid* **assignment** data has been entered by displaying error message "**Assignment** data *invalid*". 2. Actor changes **assignment** data in question. 3. Return to step 3.		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	Actor chooses one of his **assignment**s displayed by System when updating an **assignment**.		
Notes and Issues:	None		

Use Case ID:	5.3		
Use Case Name:	Delete **Assignment**		
Created By:	Wilson Lau	Last Updated By:	Wilson Lau
Date Created:	January 28, 2009	Date Last Updated:	February 09, 2009
Actors:	**Professor**		
Description:	Actor deletes an **assignment**s to System.		
Preconditions:	1. Actor is logged into System.		
Postconditions:	1. **Assignment** becomes deleted in System.		
Normal Flow:	**5.3.0 Delete a Assignment** 1. Actor requests to delete an **assignment**. 2. Actor chooses an **assignment** to delete. 3. Actor confirms to system that he wants to delete **Assignment**. 4. System deletes **assignment**.		
Alternative Flows:	None		
Exceptions:	**5.3.E.1 No Assignment to Delete** (at step 1) 1. System displays a message notifying Actor that there is no **assignment** to delete. 2. Return to step 1.		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	Actor chooses one of his **assignment**s displayed by System when deleting an **assignment**.		
Notes and Issues:	None		

Use Case ID:	5.4		
Use Case Name:	View **Assignment** *Metrics*		
Created By:	Wilson Lau	Last Updated By:	Wilson Lau
Date Created:	January 28, 2009	Date Last Updated:	February 09, 2009
Actors:	**Professor**		
Description:	Actor views **Assignment** *Metrics* for an **Assignment**.		
Preconditions:	1. Actor is logged into System.		
Postconditions:	None		
Normal Flow:	**5.4.0 View Assignment *Metrics*** **1.** Actor requests to view **Assignment** *metrics*. 2. System displays **Assignment** *metrics*.		
Alternative Flows:	None		
Exceptions:	None		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	6.0		
Use Case Name:	View **Course Grade**s		
Created By:	Jason Boyle	Last Updated By:	Jason Boyle
Date Created:	February 10, 2009	Date Last Updated:	February 26, 2009
Actors:	**Professor**, **TA**		
Description:	A **professor** or **TA** views the overall **grade** and **assignment grade**s for **student**s in a **course**.		
Preconditions:	1. Actor is logged into System. 2. At least one **course** *relevant* to Actor is not empty.		
Postconditions:	None		
Normal Flow:	**6.0 View Grade**s **for a Course** 1. Actor requests to view **grade**s for a **course**. 2. System displays the overall **grade** and **assignment grade**s for each **student** in the **course**.		
Alternative Flows:	None		
Exceptions:	None		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	6.1		
Use Case Name:	Edit **Grade**s		
Created By:	Jason Boyle	Last Updated By:	Jason Boyle
Date Created:	February 10, 2009	Date Last Updated:	February 26, 2009
Actors:	**Professor**, **TA**		
Description:	Actor edits the **grade**s for all **assignment**s of all **student**s in a **course** in spreadsheet format.		
Preconditions:	1. Actor is logged into System. 2. At least one **course** *relevant* to Actor is not empty.		
Postconditions:	The updated **grade**s are stored by System.		
Normal Flow:	**6.1 Edit Grade**s 1. Actor requests to edit **grade**s for a **course**. 2. System displays a form with input fields for **grade**s in spreadsheet format with **student**s along the y-axis and **assignment**s along the x-axis. 3. Actor enters, deletes, or modifies one or more **grade**s. 4. Actor indicates that he has finished editing **grade**s. 5. System stores the entered **grade**s.		
Alternative Flows:	None		
Exceptions:	**6.1.E.1** *Invalid* **Grade Data is Entered** (at step 4) 1. System marks the input fields containing *invalid* **grade**s. 2. Return to step 3.		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	7.0		
Use Case Name:	View **Student Grade**s		
Created By:	Jason Boyle	Last Updated By:	Jason Boyle
Date Created:	February 10, 2009	Date Last Updated:	February 26, 2009
Actors:	**Professor**, **TA**, **Student**		
Description:	Actor views his overall **grade** and his **grade** for each **assignment** in a **course**.		
Preconditions:	1. Actor is logged into System. 2. At least one **course** *relevant* to Actor exists and is not empty.		
Postconditions:	None		
Normal Flow:	**7.0 View Student Grade**s **for a Course** 1. Actor requests to view **grade**s for a **student** in a **course**. 2. System displays the **student**'s overall **grade** and **grade** for each **assignment** in the **course**.		
Alternative Flows:	None		
Exceptions:	None		
Priority:	High		
Business Rules:	EX-1		
Special Requirements:	1. A **student** will only be able to view his own **grade**s, while a **professor** or **TA** will be able to view the **grade**s of any **student** in the **course**.		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	8.0		
Use Case Name:	Set Curve		
Created By:	Jason Boyle	Last Updated By:	Jason Boyle
Date Created:	February 10, 2009	Date Last Updated:	February 26, 2009
Actors:	**Professor**		
Description:	Actor changes the curve for a **course**.		
Preconditions:	1. Actor is logged into System. 2. At least one **course** *relevant* to Actor exists.		
Postconditions:	Curve is stored by System.		
Normal Flow:	**8.0 Set Curve for a Course** 1. Actor requests to set the curve for a **course**. 2. System displays a form with an input for a weight for each **assignment** type. 3. Actor enters a weight for each **assignment** type. 4. Actor indicates that he has finished entering weights. 5. System stores the curve.		
Alternative Flows:	None		
Exceptions:	**8.0.E.1** *Invalid* **Weights are Entered** (at step 4) 1. System marks input fields containing *invalid* weights. 2. Return to step 3.		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

Use Case ID:	9.0		
Use Case Name:	View **Solution**s		
Created By:	Dana Goyette	Last Updated By:	Jason Boyle
Date Created:	January 31, 2009	Date Last Updated:	February 26, 2009
Actors:	**Student**		
Description:	Actor views the **solution** to an **assignment**.		
Preconditions:	1. Actor is logged into System. 2. A **solution** to the **assignment** exists.		
Post conditions:	1. Actor downloads a file containing the **solution** to the **assignment**.		
Normal Flow:	**9.0 View Assignment Solution**s 1. Actor requests to view the **solution**s to an **assignment**. 2. System displays **solution** files. 3. Actor selects desired file to download. 4. System downloads selected file to Actor's computer.		
Alternative Flows:	None		
Exceptions:	**9.0.E.1 No Solution Available** (at step 1) 1. System notifies Actor that no **solution** is available for the **assignment**. 2. Use case is terminated.		
Priority:	Medium		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

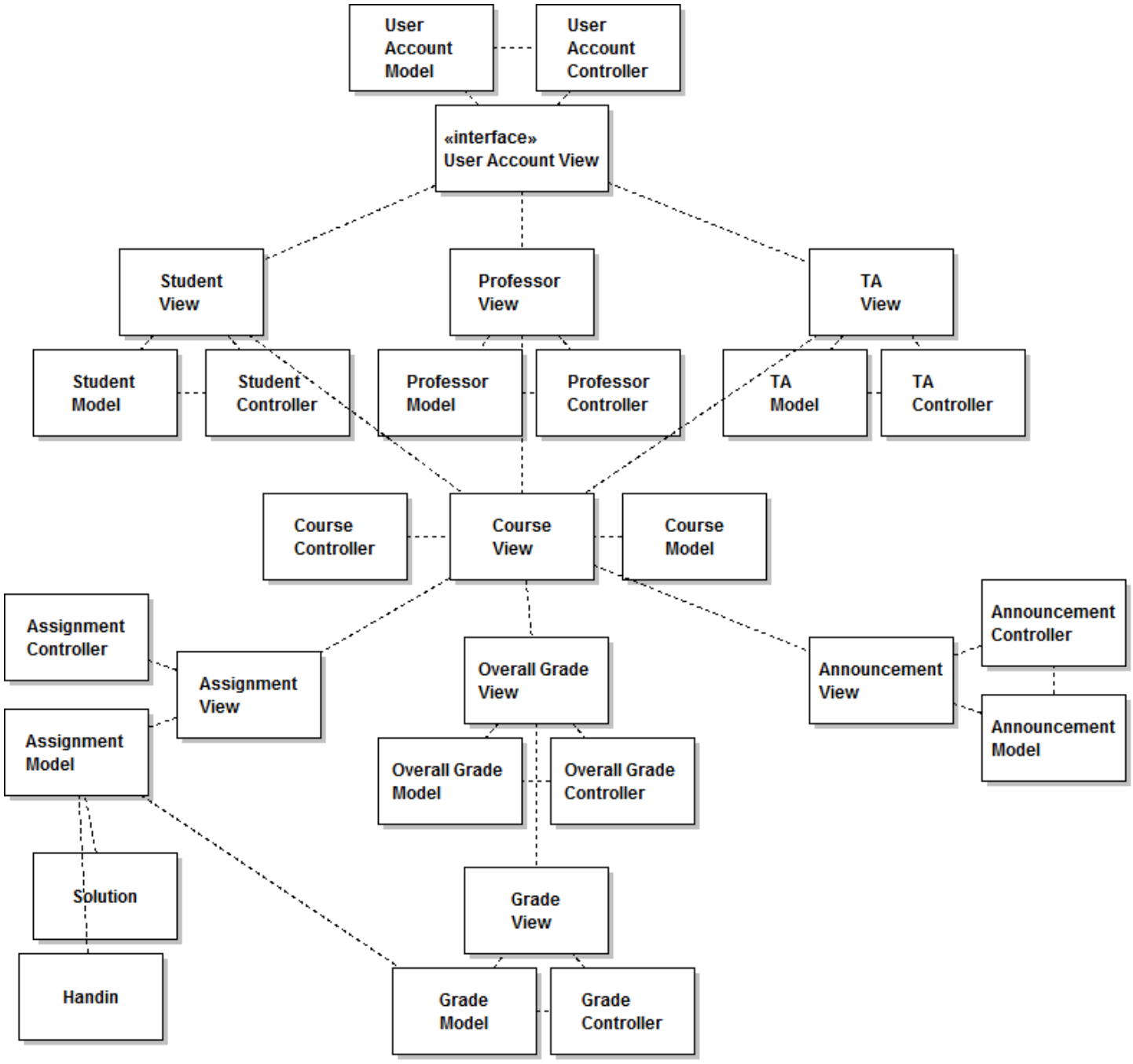
Use Case ID:	10.0		
Use Case Name:	Upload **Solution**		
Created By:	Nestor Reyes	Last Updated By:	Jason Boyle
Date Created:	February 7, 2009	Date Last Updated:	February 26, 2009
Actors:	**Professor**, **TA**		
Description:	Actor uploads a **solution** for an **assignment**.		
Preconditions:	1. Actor is logged into System. 2. At least one **assignment** *relevant* to the Actor exists.		
Post conditions:	1. System stores the **solution** files. 2. **Solution** files are accessible to all **student**s enrolled in the **course**.		
Normal Flow:	**10.0 Upload Solution** 1. Actor requests to upload a **solution** for an **assignment**. 2. System prompts Actor to select a file to upload. 3. Actor selects a file to upload. 4. System stores the selected file.		
Alternative Flows:	None		
Exceptions:	**10.0.E.2 File size exceeds file size limit** (at step 3) 1. System notifies Actor that the file exceeds the file size limit and displays the size of the selected file and the file size limit for comparison. 2. Return to step 3.		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	1. Previously uploaded **solution**s will remain available to **student**s.		

Use Case ID:	11.0		
Use Case Name:	**View Hand-In**s		
Created By:	Nestor Reyes	Last Updated By:	Jason Boyle
Date Created:	February 7, 2009	Date Last Updated:	February 26, 2009
Actors:	**Professor**, **TA**		
Description:	Actor views **Hand-In**s for an **assignment**.		
Preconditions:	1. Actor is logged into System. 2. At least one **assignment** *relevant* to Actor exists. 3. **Hand-In** has been enabled for the **assignment**.		
Post conditions:	1. **Hand-In**s are downloaded to Actor's computer.		
Normal Flow:	**11.0 View a Single Hand-In** 1. Actor requests to view **Hand-In**s for an **assignment**. 2. System displays list of **Hand-In**s uploaded by **student**s for the **assignment**. 3. Actor requests to view a **Hand-In**. 4. **Hand-In** is downloaded to Actor's computer.		
Alternative Flows:	**11.1 View All Hand-In**s(after step 3) 1. Actor requests to view all **Hand-In**s. 2. All **Hand-In**s are downloaded to Actors computer.		
Exceptions:	**11.0.E.1 No Hand-In**s **are available** (at step 2) 1. System notifies Actor that no **Hand-In**s have been uploaded. 2. Use case is terminated.		
Priority:	High		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

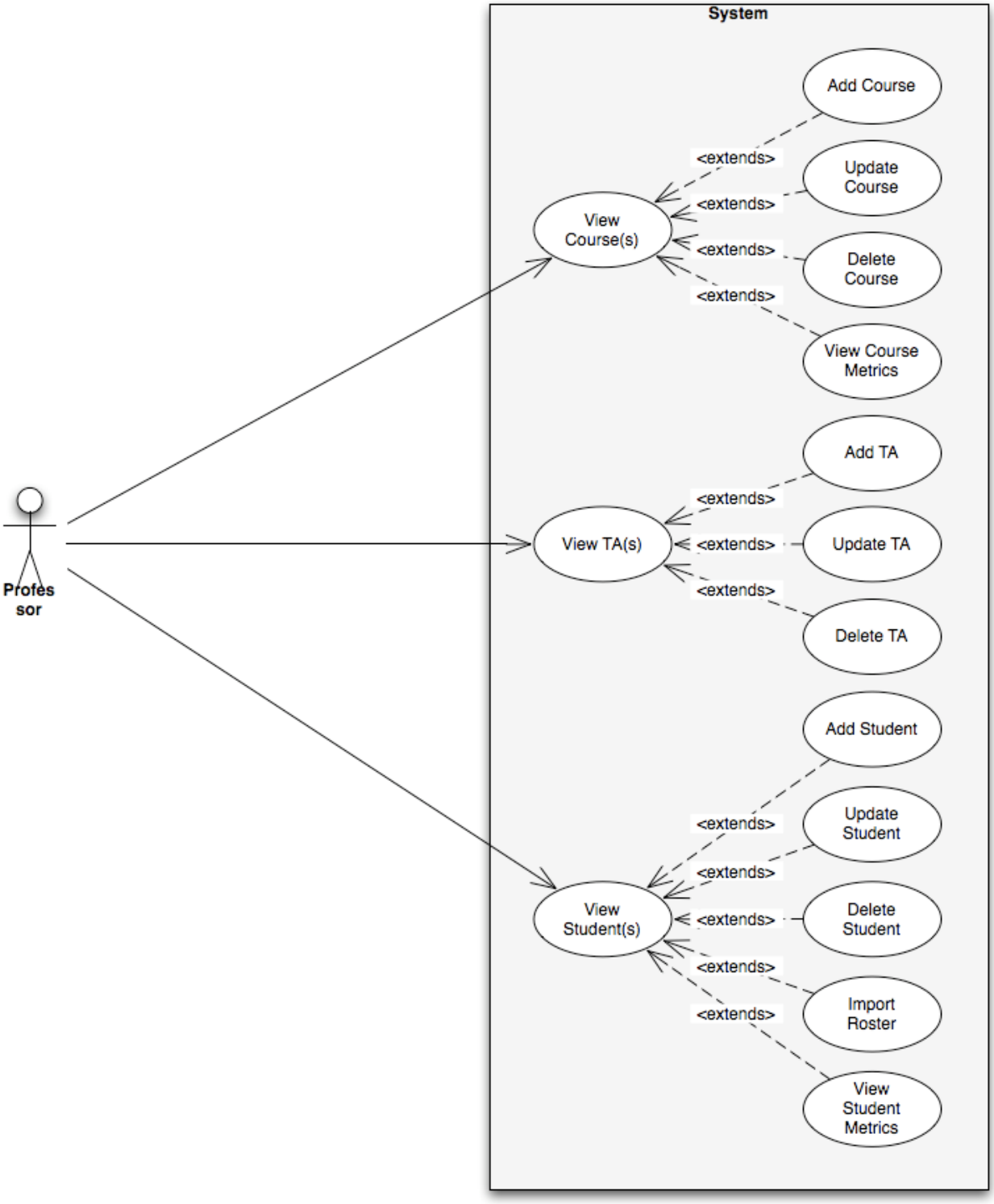
Use Case ID:	12.0		
Use Case Name:	Upload **Hand-In**s		
Created By:	Dana Goyette	Last Updated By:	Jason Boyle
Date Created:	January 30, 2009	Date Last Updated:	February 26, 2009
Actors:	**Student**		
Description:	Actor uploads a **Hand-In** for an **assignment**.		
Preconditions:	1. Actor is logged into System. 2. Actor has not received a **grade** for the **assignment**.		
Post conditions:	1. System stores the **Hand-In**. 2. **Hand-In** is available for viewing by the **Professor** or **TA** of the **course**.		
Normal Flow:	**12.0 Hand In Assignment** 1. Actor requests to upload a **Hand-In** for an **assignment** 2. Actor selects a file to upload. 3. System stores the file.		
Alternative Flows:	None		
Exceptions:	**12.0.E.1 Hand-Ins are not enabled for the assignment** (at step 1) 1. Actor is notified that **Hand-In** has not been enabled for the **assignment**. 2. Use case is terminated. **12.0.E.2 File size exceeds file size limit** (at step 3) 1. System notifies Actor that the file exceeds the file size limit and displays the size of the selected file and the file size limit for comparison. 2. Return to step 2.		
Priority:	Medium		
Business Rules:	None		
Special Requirements:	1. System shall replace an existing **Hand-In** with the latest **Hand-In** uploaded.		
Assumptions:	None		
Notes and Issues:	1. Actor may upload a **Hand-In** for an **assignment** after the due date of the **assignment**. However, the **Professor** or **TA** will be notified of how late the **Hand-In** was uploaded.		

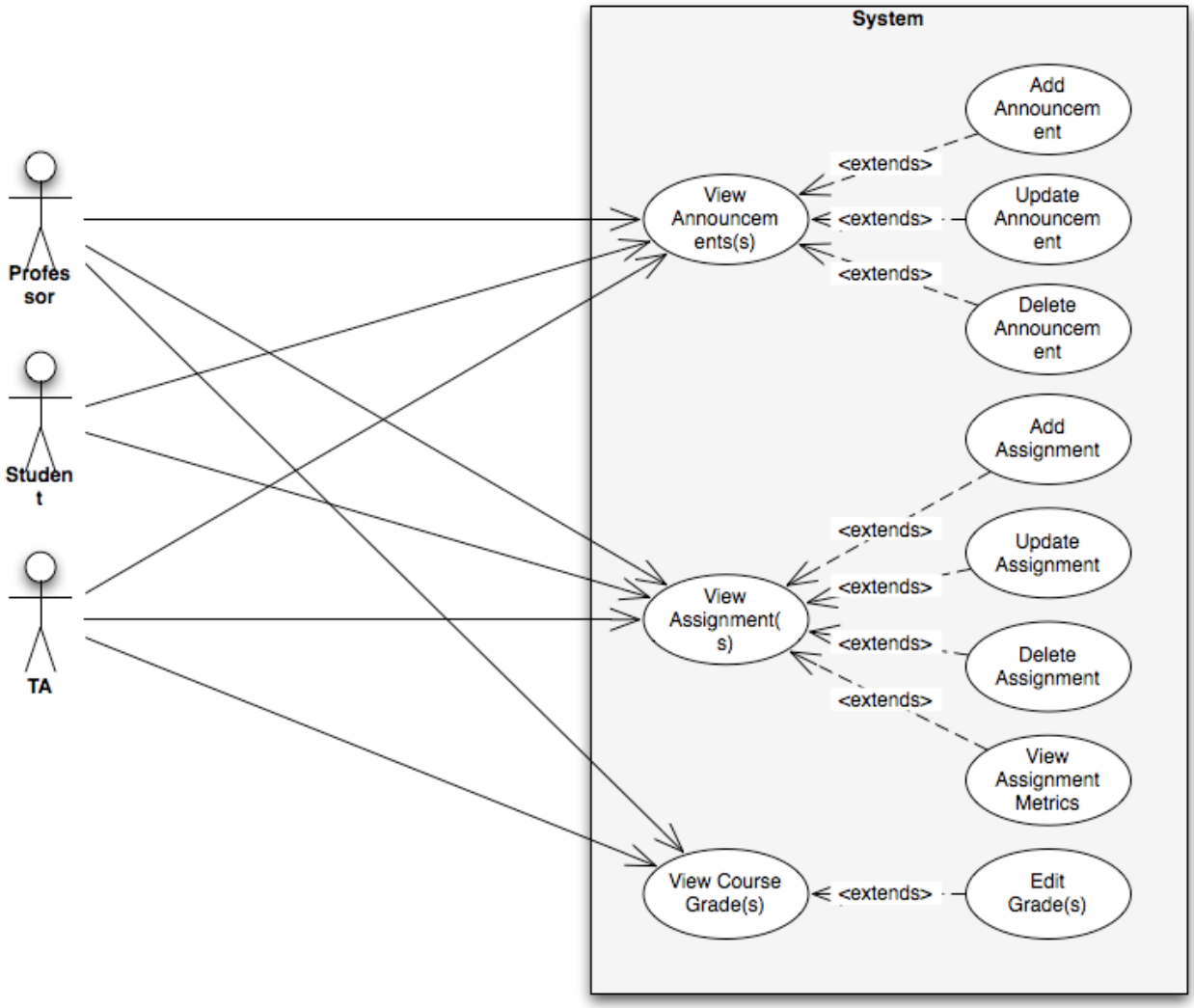
Use Case ID:	13.0		
Use Case Name:	**Grade** Estimator		
Created By:	Dana Goyette	Last Updated By:	Jason Boyle
Date Created:	January 30, 2009	Date Last Updated:	February 26, 2009
Actors:	**student**		
Description:	System displays **grade**s needed by Actor to achieve a target final **grade** in a **course**.		
Preconditions:	1. Actor is logged into System.		
Post conditions:	1. **Student**s **grade**s are not affected.		
Normal Flow:	**13.0 Estimate Needed Grades** 1. Actor requests to use the **Grade** Estimator. 2. System displays upcoming **assignment**s with an input for a theoretical score for each. 3. Actor enters theoretical **grade** for each upcoming **assignment**. 4. Actor indicates that he has finished entering theoretical **grade**s. 5. System displays theoretical final **grade** based on theoretical **grade**s input by Actor.		
Alternative Flow:	None		
Exceptions:	**13.0.E.1 No upcoming assignments exist** (at step 1) 1. Actor is notified that there are no upcoming **assignment**s for the **course**. 2. Use case is terminated.		
Priority:	Medium		
Business Rules:	None		
Special Requirements:	None		
Assumptions:	None		
Notes and Issues:	None		

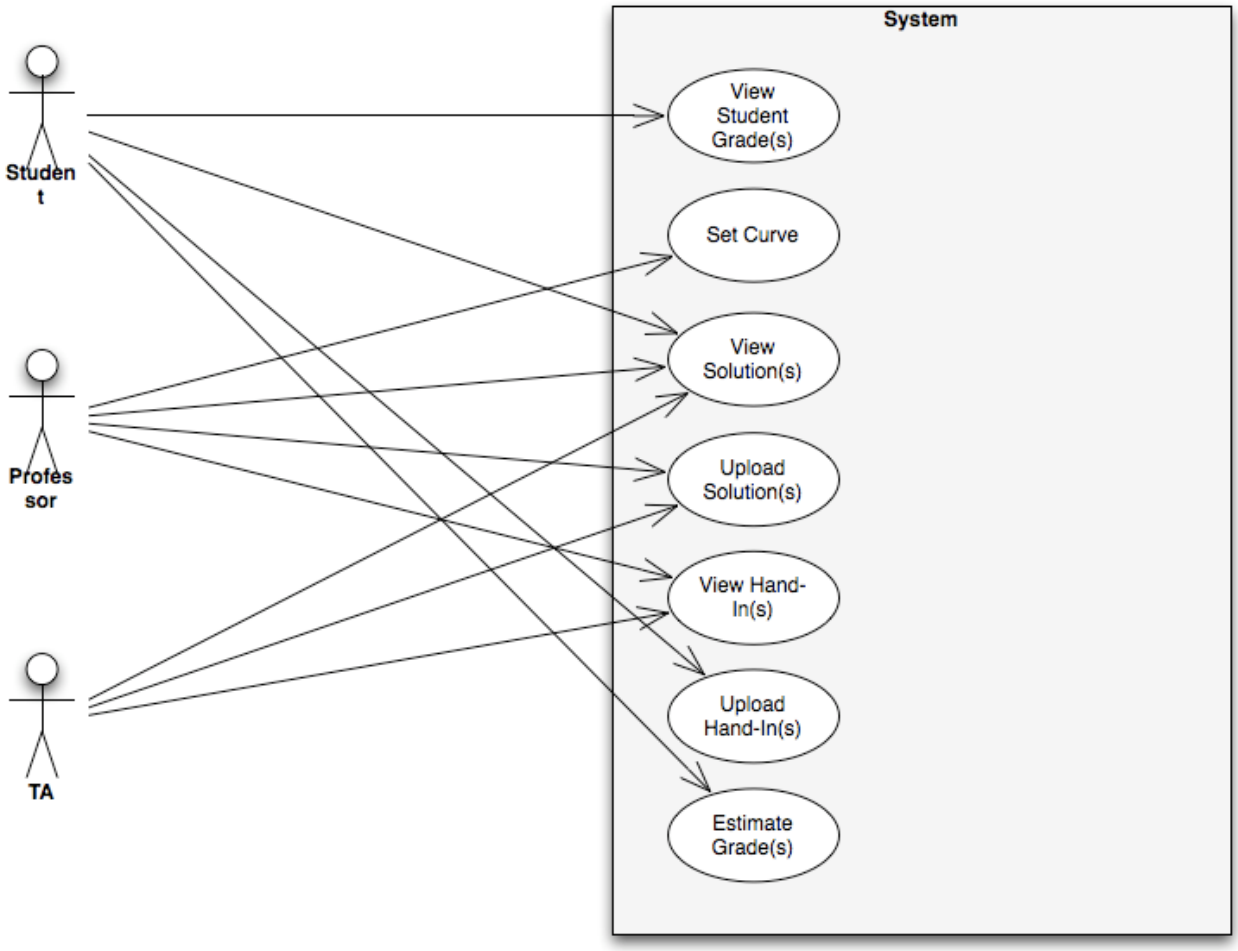
Appendix I: Class Diagram



Appendix J: Use Case Diagrams




System
Professor
Student
TA
View Announcements(s)
View Assignment(s)
View Course Grade(s)
Add Announcement
Update Announcement
Delete Announcement
Add Assignment
Update Assignment
Delete Assignment
View Assignment Metrics
Edit Grade(s)
<extends>


System
Student
Professor
TA
View Student Grade(s)
Set Curve
View Solution(s)
Upload Solution(s)
View Hand-In(s)
Upload Hand-In(s)
Estimate Grade(s)

Appendix K: Check List

General

 Cover page contains team logoand date submitted.
 Document has page numbers.
 Document matches the required format.
 Document is of professional quality.
 The credits list is specified clearly, by individual. ("done by team" is explicitly disallowed)

Consistency

 Each non-primitive entity in the data dictionary is used by one or more use cases.
 Every use case noun is in the data dictionary.
 All significant use cases are visible in the prototype.
 All views of the requirements (use cases, prototype, data dictionary) are consistent.

English

 Writing meets the standards of a the WPE level 4 or greater.
 All English, including diagram text, is written in the 3rd person using active voice, present tense verbs.
 There are no words from the bad words list.

Problem Domain

 All language is in the problem domain; don't discuss the UI!
(Exception - some non-functional requirements)
 Each view of the requirements (Use Cases, Data Dictionary, Prototype) uses the same problem-domain words.
 Excepting the prototype, there is no discussion of the UI in the SRS.

Non-functional Requirements

 Every requirement is concrete, specific, measurable and testable.
 Every requirement applies to your system.
 There are 3 maintainability requirements (to be provided by Prof. Stearns)

Data Dictionary

 The data dictionary describes all problem domain entities in the software.
 There are no glossary terms in the Data Dictionary.
 Entries are organized hierarchically and formatted appropriately.
 Every primitive (not decomposed) entity must be used:
in a composite entity
by at least one use case

Use Cases

- The proper format is used.
- Every non-primitive Data Dictionary entity is discussed in one or more use case.
- There is no discussion of the UI in the description.
- There is no textual discussion of Data Dictionary details.
- There are no design domain features described as use cases. Some common non use cases:
  a. login/logoff
  b. exit system
  c. resize window
  d. open/close window

These are system features, not use cases.

Glossary

- Every term is used in the SRS.
- There are no glossary terms in the Data Dictionary.

Prototype

- UI follows appropriate human psychology principles
- UI presentation is professional (whether a prototype or storyboards)
- UI covers all major use cases.