Boris Bojanov

AUID: 36085501

Task-Centered Design, Prototyping for GUI-Course Management System

Section 1: Tasks and Requirements

1. Introduction

Background

Athabasca University has grown and has identified the need for a system that allows teachers to manage courses, assessments and students-- taking the courses. The staff at Athabasca University have been using a command line tool, which has been used since the school first started. Their CLI (Command Line Interface) Course Management System, has some strengths that the university would like to retain as they create a GUI(Graphical User Interface) based system:

- Runs both MS-DOS X86 and Unix X86 based operating systems,
- Works with SSH(Secure Shell) tools that allow them to access machines remotely,
- Memory efficient and allows the University to extend the lifetime of their existing hardware.

Advances in technology have pushed the envelope for how humans intuitively interact with machines. Athabasca University has identified points with the CLI-system that the university intends to adopt:

- Current and future employees are expecting company software to have a GUI(Graphical User Interface).
- Diversity in staff is a strength as a teaching institution, but this poses some challenges for them when trying to train staff on their systems and processes.

The GUI-Based Course Management System aims to achieve the following goals:

Boris Bojanov

AUID: 36085502

- Reduce training time
- Prevent invalid data entry
- Support role-based workflows
- Improve navigation and discoverability

Users should be able to intuitively be able to find functionality without having to read complicated manuals.The GCM-System (GUI Course Management System) must be able to integrate with existing Database systems which rely on. The system is robust and will not allow incorrect data entries and can handle exceptions so that the user is aware of data type constraints. The GCM-System will guide users through intuition through its UI (User Interface), to aid in navigation. This also will aim to convey to the user what types of data are expected and acceptable for any given entry.

Expected types of users of the system

- Allow for the user to add courses,
- add assessments to the course,
- add students to the course,
- add grades that students achieve in each assessment.
- Read/Write course data to ".bin" so data is persistent
- The users for the system will be; Tutors who mark assignments,
- Professors who mark and create course outlines,
- Administrators who edit the students who are taking the course.
    - In the future the system will aim to allow for students to use the system to organize their semesters and plan their course registrations.

Work contexts—a description the work setting

Tutors marking assessments, will need to be able to access the course's list of assigned students and enter a grade for a specified student and assessment pair. Tutors may also need to change the grade once it has been entered.

Professors marking and creating course outlines, will need the same usability as the "Tutor" user group with a higher access level of privilege. They approve the change grade

requests from “Tutors”, or change grades themselves. Professors will also need to be able to access the list of assessments within a course and edit them. This will be part of the course outline that they create, any changes to the course outline are tracked by revision number.

Administrators edit the student list for each course. They will need to be able to assign multiple “Student” users to a course(Or assign multiple courses to one or more students). Administrators approve course outlines and make sure that they fit the school’s curriculum.

What the system will be used for.

The GCM-System will be used to create and/or edit a course. Within each course will be assessments that can be assigned weights and names, additionally courses contain students which are assigned to the course. Students may be added to the course or removed from a course. The GCM-System will be used to track which students are currently enrolled in a course and easily find student grade records. GCM-System must be able to keep student data relevant and current.

System constraints

The system must be able to work on all major operating systems (Linux/Unix, MacOS, Windows). The system must use system resources efficiently. The GCM-System must be responsive and handle errors with messages that help the user identify what they need to do differently. The system must guide users through intuition through its UI to convey to the user what the expected and acceptable data types are for any given entry.

2. Concrete Task Examples

list at least five to seven concrete task examples. The task descriptions should follow the structure of the example provided on page 9 of the first reference introduced above, Working through Task-Centered System Design (Greenberg, 2003).

Boris Bojanov

AUID: 36085504

Task 1: Change Grade

Target Users:

Primary Stakeholder**:**Professor

Other Stakeholder: Tutor, Student

Scenario:

The email from TutorY was clear when it described the necessary grade corrections for StudentA, In the email the course is clearly written, “COMP400” and the assessment name “Assignment 2” followed by the student AUID “3608550” and grade that is currently shown in the system “67%” which is to be changed to “100%”. TutorY included a short explanation for the correction but ProfessorX didn’t finish reading it when the GCM-System opened up, something to the effect of “missclick”. When ProfessorX is asked to change a grade that TutorY has mistakenly entered, the professor will open the GCM-System and enter his username “Xavier” and password “StanLee”, with this the GCM-System now knows which courses to display to ProfessorX for him to interact with. The GCM-System now also knows that this user has the privilege to change the grade of an assessment that already has been graded. ProfessorX sees a button called “List Courses” which he presses to find “COMP400”, within the course he sees the course outline, full name of the course “Computer Science 400: Computer and Network Security”, the assessments, and a button for the list of students. The list shows the student’s name and AUID, the search box allows him to enter the AUID that he is interested in and then the system folders the list to show just the students with that ID, which is luckily only one. The grade for the assessment is changed and then ProfessorX closes the session. The system saves the updated information and sends TutorY and the StudentA an automated message, informing them of the change, and the updated course grade.

Why this task matters

This task is commonly done in the course of a semester, professors and tutors alike will need to make changes to an assessment after the fact. One of the Tutor’s main roles will be to enter the grade for an assessment. This task must be done with accuracy as the student’s final grade will be impacted by these changes. Student records are based on these inputs and will be reflected in their letter grade.

Boris Bojanov

AUID: 36085505

Representativeness

The process, entering grades achieved for an assessment, happens frequently and mistakes in that process are likely to happen. The system must be able to accommodate the Tutor user group as well as give different levels of access to the student's course data, so that data remains accurate and up-to-date.

Variations

The correction requested by a tutor vs. discovered by professor vs. raised by a student inquiry.

The search by AUID vs. search by name (possible duplicate names).

A tutor may enter grades but may require professor approval to modify after submission.
Single grade correction vs. multiple corrections for the same assessment

Correction before final-grade release vs. After final grade has already been calculated and communicated.

Task 2: Add Student to multiple courses 1:N

Target Users:

Primary Stakeholder: Administrator

Other: StudentA, Professor

Scenario:

The Faculty of Science has forwarded an email from a student to *Wade Winston Wilson*, requesting to be registered in a list of courses as per their course registration plan. *Wade Wilson* is one of the critical administrators who is allowed to change the list of registered students in the catalog of courses that Athabasca University offers. When *Wade Wilson* is asked to do this they will open the GCM-System and enter their username "Vault76" and password "VaultTechDay0" for the system to be able to

Boris Bojanov

AUID: 36085506

identify their administrative role and permissions. The GCM-System also allows *Wade Wilson* to see a course list, but unlike ProfessorX's view, the list contains *all* the courses that the Faculty of Science offers to students at the University. Wade now filters for students with the name "Deadpool" that they are interested in and selects their name displayed in the list, unfortunately there are many students that go by this name. So *Wade* matches them AUID to identify the correct one, and then selects the option to "add student to course", which again presents a list of courses. Thank goodness that this list allows *Wade Wilson* to add all six courses to the student for the coming semester. All changes are saved and the Student is sent an automated message saying that they have been registered to their courses and ProfessorX can now see them in their list of students in the student list.

Why this task matters

The data needs to be trusted and consistent for multiple users. The system must be able to give different levels of access to the students assigned to a course, so that data remains accurate and uptodate. To ensure that data is not accidentally changed administrators are the ones who will have the permission to add or remove students.

Representativeness

This task represents a common administrative workflow involving a one to many relationship of changes. The administrators user group as well as the Professor user group will need access to this data, regularly.

Variations

Student identified by name vs. AUID, especially in cases of duplicate names.

Adding a student to a single course vs. adding them to several courses at once.

Enrollment before the term begins vs. late enrollment after a course has started.

Immediate vs. delayed propagation of enrollment changes to professors and students.

Boris Bojanov

AUID: 36085507

Task 3: Add New Course

Target Users:

Primary Stakeholder: Administrator

Other: Professor, Student

Scenario:

Athabasca University has decided to add a new course for students to register into. The University sends all the necessary information to the Overseer, who needs to know the course name, the abbreviated name, the professor assigned to it, the date the course will first be offered, and the assessment schedule, as a list of assessments. Each assessment item is represented by (ID, name, weight) where the ID is unique for that assessment; name is the assessment name, such as Assignment 1; and final exam. Weight is the percentage of the assessment towards the final grade. The Overseer is responsible for setting up the course properly so that the total weights equal to 100%, and that all the necessary course information is filled out accurately. This is crucial because the course will be referenced by other faculty staff and students for their use cases of the GCM-System.

Why this task matters

The system is crucial to maintaining an accurate record of the course. The system must be able to give different levels of access to the students assigned to a course, so that data remains accurate and uptodate. The administrators user group as well as the Professor user group will need access to this data. To ensure that data is not accidentally changed administrators are the ones who will have the permission to add or remove students.

Representativeness

This task represents a setup and configuration workflow that occurs regularly as programs evolve. It shows the importance of a robust course management system. During the step of adding or removing a course, many potential issues could arise. This is why its is key

to limit the permissions for this action and to ensure that the system is well prepared to handle errors.

Variations:

Courses with a small number of assessments vs. courses with many graded components.

Course creation well in advance of the term vs. late additions close to the registration deadline.

Initial course creation by an administrator vs. later edits or refinements by a professor.

Assessment weights entered incrementally vs. adjusted after all assessments are defined.

Task 4: Remove a student from a course

Target Users:

Primary Stakeholder: Administrator

Other stakeholders: Student, Professor

Scenario:

Lex Luther is a student at Athabasca University and is working part time to support his studies. His job has recently changed his hours and Lex Luther is feeling that he is not able to keep up with his course load. Lex contacts his course administrator, Alexander Joseph, and asks to drop the course MATH315. Alexander checks over the request and finds that it meets the university's policy and guidelines for dropping a course. Alexander boots up the GCM-System which has his password saved, reducing the time to login and identify himself to the system, through the options available to him, he finds “Lex Luther” in the database and selects to see the list of courses that he is associated with. The list includes MATH315 and Alexander is able to remove the course from the students list. The change is saved and the system has updated it for ProfessorX and the student. This updated the degree tracking software and now unchecks the course from the completed list.

Boris Bojanov

AUID: 36085509

Why this task matters

Athabasca University has many students and as an online university they offer a unique amount of flexibility for students. Some students may want to change the course that they are registered in or drop a course. In both of these cases the administrator responsible for the course in question must be able to properly remove a student. This ensures that the list of students is accurate and allows professors to trust the list when they need to find a student in their designated courses.

Representativeness

This task represents routine enrollment maintenance that occurs throughout a term.

Variations

Withdrawal before the course begins vs. withdrawal after assessments have been completed.

Removing a student from a single course vs. withdrawing from multiple courses.

Withdrawals allowed without penalty vs. withdrawals requiring special approval.

Updates that affect only course rosters vs. updates that also affect transcripts or degree tracking systems.

Task 5: Remove a Course

Target Users:

Primary: Administrator

Other stakeholders: Professor, Tutor, Student

Scenario:

Lois Lane is an administrator who is responsible for maintaining the list of active courses. Her role also requires her to create and analyze attendance records for

Boris Bojanov

AUID: 360855010

each course. Lois Lane has noticed a distinct drop in signups for the “BEEP201 Morse Code course” and at the end of the term the course will need to be removed from the active list. The course contains assessments, students, tutors, and an Instructor/s. When Lois Lane removes the course from the active list each of the user groups should be able to see the updated information as it pertains to their role in the course management system. She is working remotely from her office computer using the GUI-Based Course Management System and is performing this task during regular administrative hours. Lois is an experienced administrator who is comfortable with computers but is not technically inclined and does not have knowledge of database systems or file storage formats. She expects the system to guide her through the process and to clearly indicate the consequences of removing a course before the action is finalized.

Why this task matters

Sometimes the field of study changes its scope and the required courses to prepare students for their professional career changes. An appropriate professor may no longer be available to teach the course that is currently being offered. Demand from students may be too low to justify offering a course. These are all hypotheticals demonstrating the need for being able to remove a course. This necessitates the action for a course to be removed from the catalog and ensuring that the data is accurately updated in all critical areas.

Representativeness

This task represents infrequent but critical maintenance of the course catalog.

Variations

Course removal due to low enrollment vs. curriculum changes or staffing limitations.

Removal at the end of a term vs. cancellation before a course begins.

Complete deletion vs. archival or historical retention of course data.

Removal affecting only future offerings vs. removal impacting current or recently completed students.

Boris Bojanov

AUID: 360855011

3. Tentative list of requirements

Tentative list of requirements. From the task examples, extract the major system requirements and prioritize them by a) absolutely must include, b) should include, c) could include, and d) exclude. Each category should be accompanied by a discussion of why items were placed in that category. The list reflects the needs of the prospective users, administrators, professors, tutors, and students who interact with the GCM-System under varying levels of responsibility and risk. Because it is unrealistic to support all possible requirements equally, they have been prioritized according to their importance to the successful operation of the system.

a) absolutely must include

- Role based access control (permissions)
- Create, edit, remove courses
- Add or remove students from courses
- Enter and modify assessment grades
- Assessment weight validation and calculation (total equals 100%, calculate final grade based on weighted marks)
- Persistent storage of course data
- Clear error handling and confirmation for destructive actions

The system must distinguish between administrators, professors, tutors, and students, and restrict actions accordingly. Tasks such as changing grades, adding or removing students, and removing courses all rely on different privilege levels to prevent unauthorized or accidental data modification. Administrators must be able to add new courses and remove courses from the active catalog. Without this functionality, the system would fail to support the university's academic lifecycle. Managing student enrollment is a core administrative responsibility demonstrated in Tasks 2 and 4. The system must maintain accurate and current enrollment records to support grading, reporting, and curriculum tracking. Tutors and professors must be able to enter grades and correct mistakes. Because grades directly impact students' academic records, this requirement is fundamental. The system must ensure accurate final grade calculations and prevent invalid course configurations. The system must reliably load and save course data, including students, assessments, and grades, using persistent storage (.bin files or Databases). Every task assumes that data changes are retained across sessions. High-risk tasks such as removing students and removing

courses require clear warnings and confirmations so users understand the action before proceeding.

b) should include

- Search and filtering capabilities
- Ability to audit critical changes
- Notifications for relevant stakeholders when changes are made that impact them
- Revision tracking for course outlines

Tasks 1 and 2 demonstrate the need to locate students or courses using identifiers such as AUIDs or names. Search functionality reduces errors and improves efficiency, especially in large datasets. Tasks involving grade changes and course modifications would benefit from tracking who made changes and when. This supports accountability and institutional trust, especially in disputed grading Scenarios. Notifying students and tutors of grade changes and enrollment changes helps keep users informed and reduces manual communication overhead. Task 3 implies that course outlines may change over time. Tracking revisions helps professors and administrators manage updates without confusion.

c) could include

- Student-facing access to view grades and course status
- Batch operations for administrators
- Integration with external degree-tracking or scheduling systems

While students are identified as future users, none of the core tasks require direct student interaction. Providing read-only access to grades and enrollment status could be added later. Bulk enrollment or removal of students (e.g., across multiple courses) would improve efficiency but can be deferred without impacting core functionality. Task 4 references updates to degree tracking software. Full integration would be beneficial but introduces additional complexity and dependency risks.

d) exclude

Boris Bojanov

AUID: 360855013

- Real-time collaboration between users
- Advanced analytics or predictive enrollment features
- Customization of interface themes or layouts

None of the tasks require simultaneous multi-user editing. Supporting real-time collaboration would significantly increase complexity without clear task-driven justification. Although potentially useful, no tasks indicate a need for analytics beyond basic record management. While aesthetics may improve user satisfaction, none of the tasks depend on visual customization, making this a low-priority feature. All of these would be good to add but would not significantly add to the usability and goals of the system.

4. Concluding Recommendation

Based on the task-centered analysis conducted for the GUI-Based Course Management System (GCM-System), we found that it is recommended that Athabasca University proceed with the development of the system using an iterative, task-driven design approach. The task examples demonstrate that the current command-line system (CLI-System) no longer adequately supports the needs of a diverse user population performing high-responsibility tasks (i.e. as managing grades, enrollments, and course records). A GUI-based system designed around the administrative, instructional, and academic workflows has the potential to significantly reduce training time, and user frustration. With refinement the system could aid in reducing errors, improving data accuracy and confidence in the system.

The task analysis shows the importance of prioritizing core functionality in development. Features supporting role-based access control, flexible course and enrollment management, grade entry and correction, and persistent data storage should be implemented first, as they directly support critical and high-risk tasks performed by administrators, professors, tutors, and students. Lower-priority features, such as student access or external system integrations, should be deferred until the core system has been validated.

The system has some notable barriers to development. One major challenge will be managing the transition from the existing command-line system to a graphical interface, particularly for staff who are accustomed to legacy workflows. Resistance to change and the need for user training may impact adoption if not addressed through intuitive design and gradual rollout. A collaborative effort with the CLI-system team could aid in transitioning

Boris Bojanov

AUID: 360855014

from the GUI-System system to CLI-system. Data migration from existing systems also presents a risk, historical records must be preserved accurately to maintain institutional trust. Lastly, cross-platform compatibility and maintaining efficient system performance will require careful design decisions, especially given the university's reliance on diverse hardware and operating environments.

Despite these challenges, the task-centered design process provides a strong foundation for mitigating risk by grounding development decisions in real user needs rather than assumed functionality. Proceeding with basic prototyping and imagined task walkthroughs will allow the university to identify usability issues early and refine the system before committing to full implementation. Lastly creating an efficient development cycle loop will allow for users to give feedback that can be interpreted and implemented with regular maintenance updates.

Section 2: First Prototype and Walkthrough

Prototype

At least one storyboard or sketch for each requirement

- Role based access control (permissions)
- Create, edit, remove courses
- Add or remove students from courses
- Enter and modify assessment grades
- Assessment weight validation and calculation (total equals 100%, calculate final grade based on weighted marks)
- Persistent storage of course data
- Clear error handling and confirmation for destructive actions

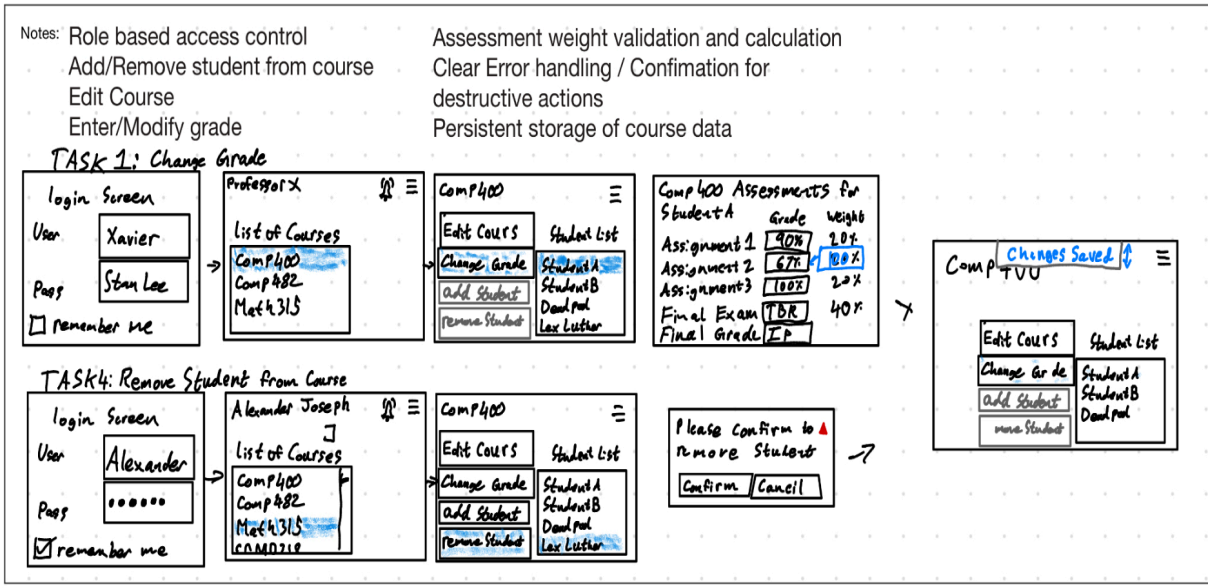
Prototype A: Course Centric Workflow

start at course -> student -> assessment

Boris Bojanov

AUID: 360855015

Prototype A assumes that most work begins with a course context. After login, users see a course list appropriate to their role (e.g., professors see their courses; administrators see all courses). Selecting a course reveals course details and exposes actions such as managing assessments, viewing the student roster, and performing grade changes. This design prioritizes efficiency for tasks that are repeated across many students inside a single course.



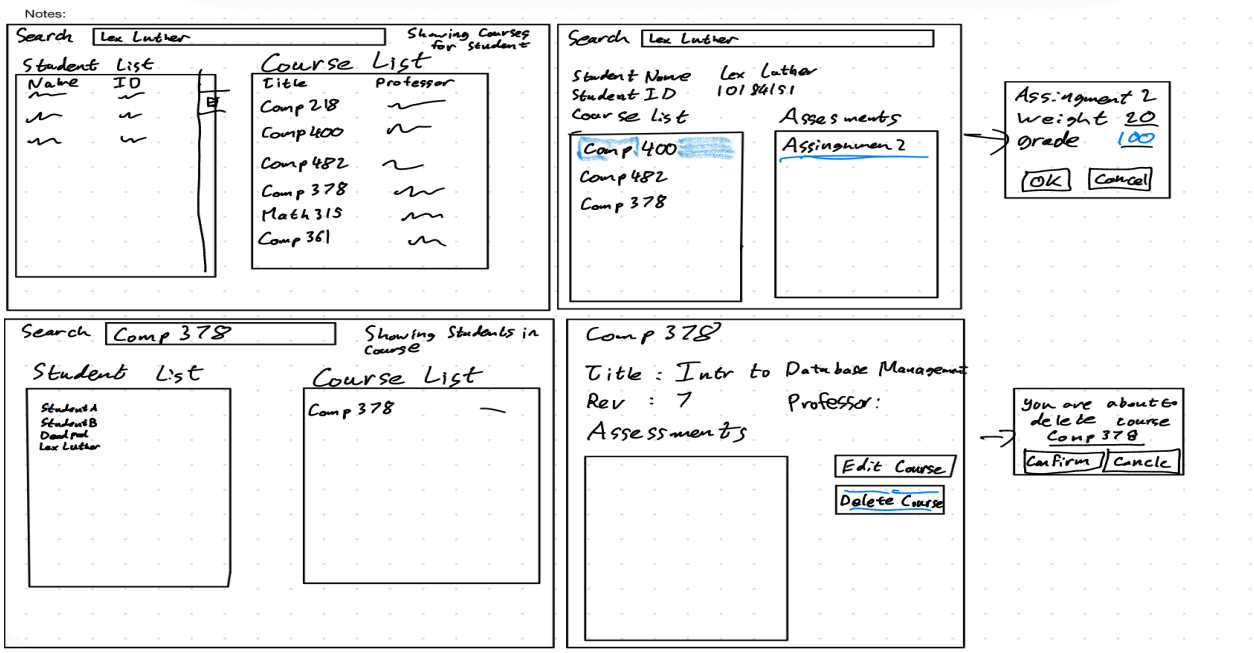
Prototype B: Student-centric workflow

start from student-> course-> assessment

Prototype B assumes that many requests originate from a specific student (e.g., grade disputes, enrollment changes, special cases). After login, users begin by locating a student, then view the student's enrolled courses and associated assessments. Actions are enabled only after a valid student is selected, which emphasizes error prevention for high-risk tasks (grade changes, enrollment changes). This design prioritizes accuracy and speed when tasks start from a student identity.

Boris Bojanov

AUID: 360855016



Task Centered Walkthrough

Task 1: Change Grade Walkthrough

User Context:

This walkthrough evaluates how ProfessorX changes a student's grade using Prototype A: Course-centric workflow. ProfessorX is an experienced faculty member correcting a grading error submitted by a tutor.

User Intent:

Change StudentA's Assignment 2 grade for COMP400

Boris Bojanov

AUID: 360855017

Task 2: Add Student to course

User Context:

Wade Wilson (Administrator) must enroll Lex Luther into COMP218. This is also a high-risk task because it changes enrollment records and affects both administrative reporting and instructional access.

User Intent:

Wade Wilson needs to add a student “Lex Luther” to a course “COMP218”.

Prototype A - Task 1: Change Grade Walkthrough

Walkthrough Steps

Step 1 -- Locate the Relevant Course

User Action: ProfessorX logs into the system and is presented with a list of courses that he teaches. He selects COMP400 from the course list.
System Response: The system displays the course details, including the course title, assessment options, and the list of enrolled students.
Evaluation: The system successfully supports the user’s goal by presenting only courses relevant to ProfessorX, reducing the risk of selecting an incorrect course.

Step 2 -- Select the Student

User Action: ProfessorX selects the “Change Grade” option and chooses StudentA from the list of enrolled students.
System Response: The system displays a list of assessments associated with StudentA, along with the current grades for each assessment.
Evaluation: The system provides the necessary information to proceed with the task, but requires the user to visually scan the student list to locate the correct student.

Boris Bojanov

AUID: 360855018

Step 3 -- Modify the Assessment Grade

User Action: ProfessorX selects Assignment 2 and enters the corrected grade value.
System Response: The system accepts the new grade and presents a confirmation message indicating that the change has been saved successfully.
Evaluation: The confirmation feedback clearly indicates task completion and reassures the user that the modification has been applied.

Evaluation:

Prototype A supports the professor's goal efficiently when the correction is anchored to a known course. The system minimizes navigation and reduces repetition of actions. Locating StudentA requires scanning a roster unless the design includes robust search/filtering by AUID. In large courses, roster scanning increases time and increases the chance of selecting the wrong student.

Implications for Redesign

The walkthrough suggests that while the course-centric workflow is effective for course-focused grading tasks, it may be less suitable for situations that originate from a student-centric perspective. Add AUID-based search/filtering and make it visible and fast.

Prototype A - Task 2: Add Student to course

Walkthrough Steps

Step 1 -- Wade selects the course course list.

User Action: Wade selects the course COMP218 from the course list.
System Response: The system displays the course information, including assessment details and the current list of enrolled students.
Evaluation: The System successfully aids the user in finding the students that exist in the system.

Step 3 -- Find a student

User Action: Wade selects the option to add a student to the course.
System Response: The system presents a search interface that allows Wade to locate a

Boris Bojanov

AUID: 360855019

student.
Evaluation: The user is required to visually locate the student that is being searched.

Step 4 -- Select the student

User Action: Wade searches for the name “Lex Luther” and uses the student’s AUID to identify the correct individual from the results.
System Response: Wade selects the student and confirms the enrollment action when prompted by the system.
Evaluation: The system has successfully identified the student that the user intends to manage for this course

Step 5 -- add a student

User Action: Wade selects the student and confirms the enrollment action when prompted by the system.
System Response: The system prompts the user with the option that confirms the enrollment action.
Evaluation: The student has been added to the course and the system saved the changes.

Evaluation:

Prototype A fits a course-centric administrative workflow well. It minimizes the chance of adding the student to the wrong course because the course is chosen first. The primary usability risk is the student identification step. Searching by name and verifying by AUID is simple, but the design must support disambiguation clearly (showing AUID). The workflow assumes that the administrator already knows which course the student should be added to, which may not always be the case.

Implications for Redesign

The walkthrough suggests that student search and identification could be further optimized, particularly for large student populations. Providing additional filtering or clearer differentiation in search results may reduce cognitive effort and improve efficiency. Future iterations may also explore alternative workflows that allow administrators to begin enrollment tasks from a student-focused perspective when appropriate.

Boris Bojanov

AUID: 360855020

Prototype B - Task 1: Change Grade Walkthrough

Walkthrough Steps

Step 1 — Locate the Student

User Action: ProfessorX logs into the system and is presented with a list of students. He uses the search box to find StudentA (e.g., by name or AUID).

System Response: The system filters the student list to show matching results. When StudentA is selected, the system displays StudentA's profile context and populates the course list showing the courses StudentA is registered in.

Evaluation: The system supports the user's goal by letting the task begin with the student, which matches real Scenarios where grade corrections are requested using a student identifier. The search box reduces manual scanning effort compared to selecting from a long list.

Step 2 — Select the Relevant Course

User Action: ProfessorX selects COMP400 from StudentA's course list.

System Response: The system filters the assessment list to show only assessments associated with COMP400 for StudentA.

Evaluation: Filtering assessments by course helps prevent errors by narrowing the user's choices to the correct course context. This supports accuracy and reduces the risk of modifying the wrong student record.

Step 3 — Modify the Assessment Grade

User Action: ProfessorX selects "Assignment 2" and enters the corrected grade.

System Response: The system accepts the new grade, updates the displayed value, and shows a confirmation message indicating the change has been saved.

Evaluation: Clear confirmation feedback is appropriate for a high-risk action and provides assurance that the correction was successfully applied.

Boris Bojanov

AUID: 360855021

Evaluation:

The Prototype B works well for allowing ProfessorX to quickly access all relevant information for a specific student without needing to navigate through multiple courses. This design also reduces cognitive load for "find the student" tasks because search is integrated into the starting screen.Filtering assessments by course after selecting the student reduces the likelihood of modifying the wrong record and helps maintain accuracy.

Implications for Redesign

Prototype B should remain available for student initiated corrections, but it should be paired with a course-centric mode for course-wide work. The interface should also visually represent what is being filtered or searched ("Assessments for COMP400") for clarity.

Prototype B - Task 2: Add Student to multiple courses

Walkthrough Steps

Step 1 — Log in and arrive at student list

User Action: Wade logs in.
System Response: The system shows a searchable Student List; the Course List panel is empty or shows "Select a student to view courses."
Evaluation: Good starting state. This matches the student-centric model and reduces early errors (can't change enrollment without a selected student).

Step 2 — Search for the student by name

User Action: Wade types "Lex Luther" into the search box.
System Response: Student List filters to show all matching "Lex Luther" entries, each with a visible AUID.
Evaluation: Supports the task well because name search is fast and AUID disambiguation is available.

Boris Bojanov

AUID: 360855022

Potential usability risk: If many results appear, Wade may still need extra filtering (e.g., program, DOB, email) to reduce scanning.

Step 3 — Select the correct student using AUID

User Action: Wade clicks the Lex Luther entry with the correct AUID.
System Response: The system highlights the selected student and populates the Course List with currently enrolled courses for that student. The "Add student to course" button becomes enabled.
Evaluation: Strong error prevention: the system only enables enrollment actions after a valid student is selected.
Potential usability risk: The UI must make it obvious that the course list currently shown is enrolled courses, not all courses.

Step 4 — Initiate enrollment change

User Action: Wade clicks "Add student to a course."
System Response: The course panel changes to show all available courses (or opens a dedicated "Add Course" chooser). The system clearly indicates mode: "Available Courses" vs "Enrolled Courses."
Evaluation: This is the most failure-prone transition. If the panel changes without clear labeling, Wade may not realize the list meaning changed. If the button opens a separate dialog, that may reduce confusion.

Step 5 — Select the target course(s)

User Action: Wade finds and selects COMP378 (and optionally multi-selects additional courses if supported).
System Response: The system visually confirms selected courses (highlight/checkmark) and shows an action prompt (e.g., "Add selected course(s)?").
Evaluation: Works well if selection feedback is clear.
Potential usability risk: Without search/filter in the course list, scrolling could be slow in large catalogs.

Step 6 — Confirm the change

User Action: Wade clicks "Confirm."
System Response: A confirmation dialog appears: "Add Lex Luther (AUID ####) to

Boris Bojanov

AUID: 360855023

COMP378?" with Confirm/Cancel.
Evaluation: Appropriate safety step for administrative changes. This reduces accidental enrollment edits.

Step 7 — Save and show outcome

User Action: Wade confirms.
System Response: The system saves the enrollment change, shows a success message (e.g., "Enrollment updated"), and updates the Enrolled Courses list to include COMP378.
Evaluation: Strong closure: clear feedback + visible result.
Potential improvement: Provide an "Undo" option for a short time window, or show an audit log entry.

Evaluation:

Prototype B is tailored for: administrators often receive requests for a student. Enabling enrollment controls only after selecting a valid student is an error-prevention pattern that reduces accidental edits. Switching the course list from "currently enrolled courses" to "all available courses" may cause temporary confusion if the distinction is not visually clear to the user.

Implications for Redesign

Clearer visual differentiation between enrolled and available courses. When adding a student to multiple courses, the workflow may require repeated interactions with the course list, which could become inefficient for bulk enrollment tasks.