Team Outlaws

Requirements Specification

Project Sponsor and Mentor:

Dr. Eck Doerry

Team Members:

Quinn Melssen
Liam Scholl
Max Mosier
Dakota Battle

December 7, 2021

Version 2.0

Accepted as baseline requirements for the project:

Client Signature:_______________________________Date: ______________

Team Lead Signature: __________________________Date: ______________

Table of Contents

1 Introduction

As Agile programming practices continue to take the tech industry by storm, the importance of small teams in real world engineering workplaces is increasing tenfold. According to Goremotely.net, over 71% of tech companies either already use, or are in the process of adopting agile methods, which are based off of these teams. Given the prevalence of small team workgroups in the professional world begs the question, why are more engineering classes in higher education not team-based as well? A main reason for this is the difficulty for faculty to manage and maintain the teams involved in such an undertaking.

Our client Dr. Doerry has spent the last 15 years working to perfect the Northern Arizona University's computer science capstone, and as such has dealt with many of these difficulties first hand. Every year Dr. Doerry must painstakingly gather and communicate with enough clients to provide projects for the year. This process consists of hours of back-and-forth emails between many different potential clients. Once the projects have been gathered and finalized after dozens of drafts, students are expected to fill out preference documents. These are ultimately used to assign them to their respective projects, along with other information such as their GPA. This process too, requires a high amount of hands on effort that could be streamlined by a successful technology. Once the projects have a team, Dr. Doerry will then be responsible for running the capstone course and managing the collection of various assignments. Once broken down, there are three primary phases of this process, each involving large amounts of hands-on effort:

- **Gathering projects** - Involves reaching out to dozens of potential clients, exchanging hundreds of emails, and keeping track of the varying stages of each potential project.
- **Forming teams** - Involves taking in priorities, GPAs, and other information about We'd love to hear your thoughts and suggestions on how we can make TeamBandit the best it can be!each student by manually inputting all relevant information into an algorithm.
- **Executing class** - Involves using several different modes of communication to run a program, including email, websites, and verbal/written communication.

While this process ultimately leads to a successful capstone, Dr. Doerry has been brought face to face with its inefficiency as he has attempted to split leadership of the course with another NAU computer science professor, Dr. Michael Leverington. During this transitional process, Dr. Doerry has realized that his current solution could be streamlined both for himself and future instructors, as well as anyone who needs to run a team based course or project. This is where TeamBandit will come in. The intent of TeamBandit is to act as a web application that will allow Dr. Doerry and potentially any team manager to organize, collect information, and manage tasks in a centralized location. Features of this web application will include:

- A client acquisition and communication module in order to facilitate the process of finalizing project propositions.
- An assignment module that allows students to select project preferences, as well as sort them into groups using a sorting algorithm.
- A localized hub for work or deliverable submissions for both students and teachers.

This document will outline the requirements of the TeamBandit software as defined by the development team and client. This includes functional and performance requirements, environmental constraints, as well as any potential risks that may arise throughout development.

2 Problem Statement

The concerns addressed above fit into four separate modules, each one occurring chronologically throughout the semester. This has allowed our team to isolate many instances in Dr. Doerry's workflow where menial tasks can be cut out to save him time or simplify the process as a whole. As a result of this separation of concerns we have been able to implement the flowchart below, which highlights some of these problem areas and estimates the amount of time spent on each in order to better allow us to diagnose the issues and implement changes to fix them.

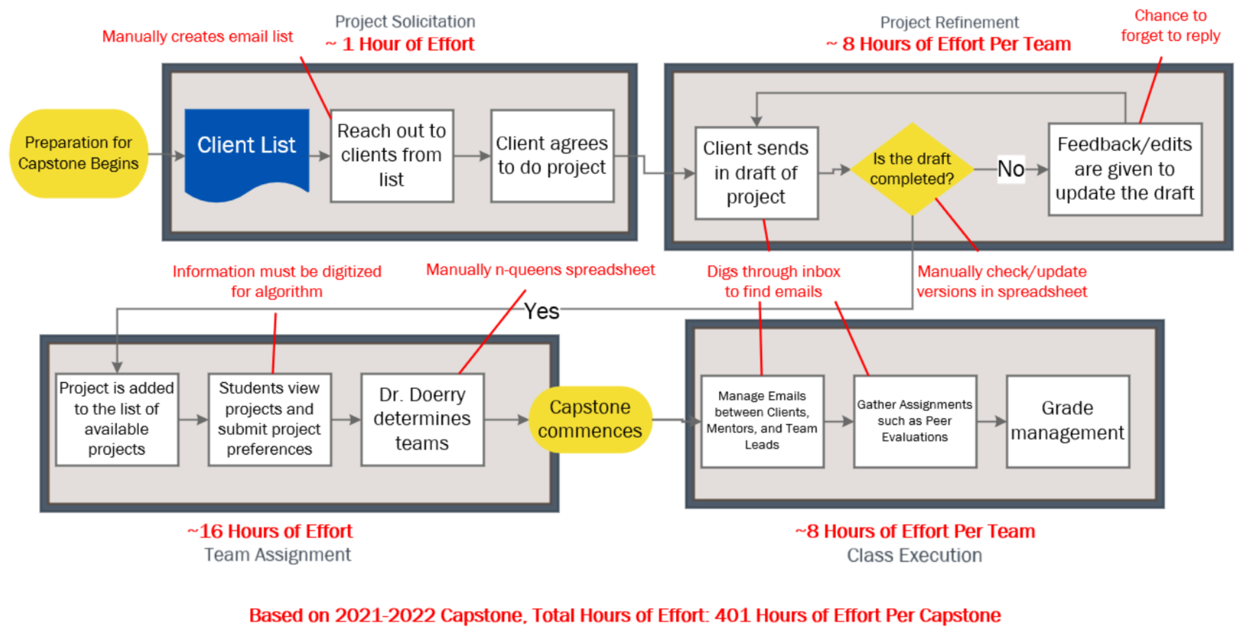
Manually creates email list
Project Solicitation
~ 1 Hour of Effort
Project Refinement
~ 8 Hours of Effort Per Team
Chance to forget to reply
Preparation for Capstone Begins
Client List
Reach out to clients from list
Client agrees to do project
Client sends in draft of project
Is the draft completed?
No
Feedback/edits are given to update the draft
Information must be digitized for algorithm
Manually n-queens spreadsheet
Digs through inbox to find emails
Manually check/update versions in spreadsheet
Yes
Project is added to the list of available projects
Students view projects and submit project preferences
Dr. Doerry determines teams
Capstone commences
Manage Emails between Clients, Mentors, and Team Leads
Gather Assignments such as Peer Evaluations
Grade management
~16 Hours of Effort
Team Assignment
~8 Hours of Effort Per Team
Class Execution
Based on 2021-2022 Capstone, Total Hours of Effort: 401 Hours of Effort Per Capstone

Figure 2.0: Diagram showing Dr. Doerry's current workflow, along with annotated problem areas and the amount of effort incurred by each.

As shown, there are several portions inside of each module that act as unnecessary time sinks. Starting the semester is often the most laborious part of the capstone experience, and can involve several manually intensive tasks such as:

- Manually creating email lists of potential clients every year
- Manually assigning students to projects based on their project preference memos
- Digging through inboxes for emails regarding project creation and finalization

The troubles do not end here however, as Dr. Doerry moves from the first three introductory modules into the fourth, his workload only seems to increase as he must:

- Manage communications between himself and his mentors/team leaders
- Manually manage submissions and check project websites across different URLs

These simple tasks manage to take hours of hands-on effort from our client, taking away valuable time that could be better spent on other facets of the capstone experience. Many of these are not limited to running college courses, and are problem areas for any institution looking to implement a team based architecture.

3 Solution Vision

Our discussion in the previous section made it clear that organizing team courses is a very arduous task due to the amount of small tasks that must be performed by hand. TeamBandit, our proposed solution for this problem, is a web application that seeks to cut out many of the menial tasks associated with hosting team-based courses. While certain areas of this process are outside of the scope of this application and have a time cost associated with things other than manual labor, TeamBandit will provide many capabilities to minimize time spent on tasks that are automatable, such as:

- Upload delimited files, such as a comma separated values (CSV) file, with relevant metadata (student IDs, client emails, etc)
  - Cuts down on labor associated with manually digitizing data
- Manage projects in the course
  - Cuts down on labor associated with checking multiple places for submissions and other relevant information
- Store client, mentor, and team lead emails in a message dashboard
  - Eliminates the need to dig through inboxes searching for certain emails
  - Eliminates the chance of waiting too long to reply or receive a reply (via indicator making it clear which party is causing the delay)
- Assign students to projects
  - Associate students to their respective projects after they have been assigned

Detailed functional elements of our solution are shown highlighted in the figure below, each corresponding to time being saved in their respective modules.

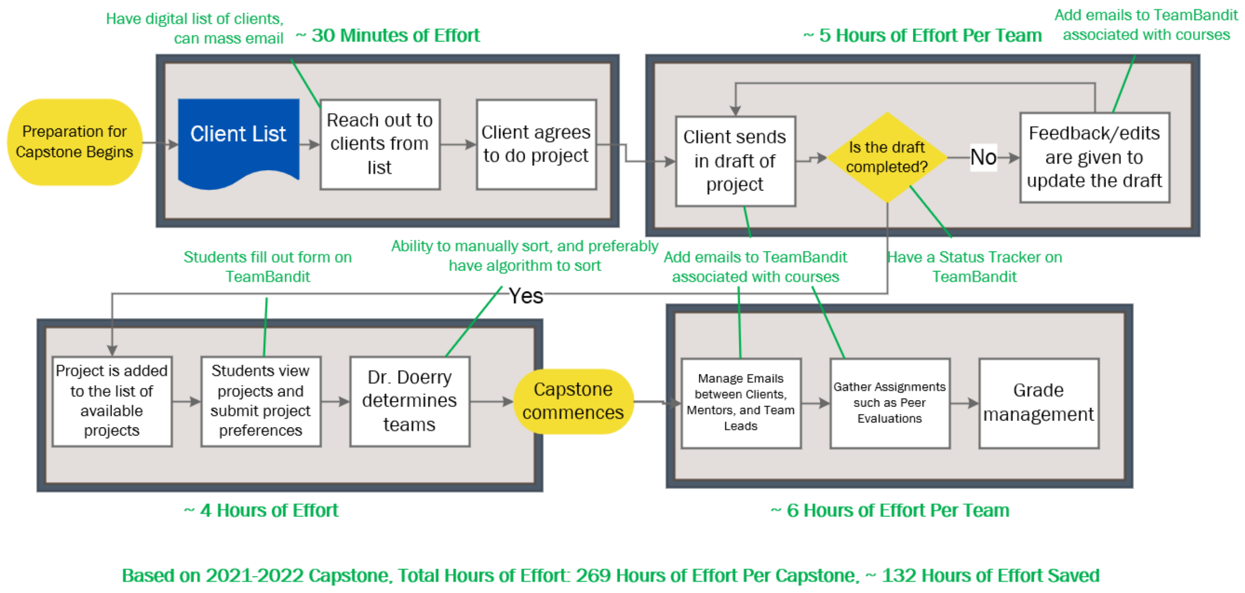
Have digital list of clients, can mass email
~ 30 Minutes of Effort
~ 5 Hours of Effort Per Team
Add emails to TeamBandit associated with courses
Preparation for Capstone Begins
Client List
Reach out to clients from list
Client agrees to do project
Client sends in draft of project
Is the draft completed?
No
Feedback/edits are given to update the draft
Students fill out form on TeamBandit
Ability to manually sort, and preferably have algorithm to sort
Add emails to TeamBandit associated with courses
Have a Status Tracker on TeamBandit
Yes
Project is added to the list of available projects
Students view projects and submit project preferences
Dr. Doerry determines teams
Capstone commences
Manage Emails between Clients, Mentors, and Team Leads
Gather Assignments such as Peer Evaluations
Grade management
~ 4 Hours of Effort
~ 6 Hours of Effort Per Team
Based on 2021-2022 Capstone, Total Hours of Effort: 269 Hours of Effort Per Capstone, ~ 132 Hours of Effort Saved

Figure 3.0: Diagram showing Dr. Doerry's envisioned workflow utilizing TeamBandit.

This diagram shows the amount of time that can be saved in each module by eliminating much of the manual labor associated with them. TeamBandit seeks to condense all of the inner workings of the capstone experience into one highly usable dashboard. Workflow revolving simple tasks such as emailing clients will go from a complicated mess involving scanning inboxes and remembering who sent what, and when, to a simple and repeatable step by step process such as:

1) Log in to TeamBandit
2) Navigate to the email hub via the simple dashboard
3) Navigate through chronologically ordered client message chains (similar to those seen on a phone)
4) Click on designated client to open up a full log of all exchanged messages
5) Clicking button to open email application, where one may send an email and have it quickly logged back into the message chain

By moving all of the working parts into one application it also limits the possibility of things going awry or flying under the radar, such as clients or teams who have not communicated with the professor in a long period of time.

Ultimately, college capstone courses are not the only organizations suffering from these inefficiencies, and TeamBandit will provide a way for academic and business institutions to quickly and effectively run team based projects, enhancing their overall workflow.

4 Project Requirements

The next part in the design process is evaluating the certain requirements the product must fulfill. The requirements that will be discussed are:

- Functional Requirements - A specification defining what the product should be able to do. As an example, what functions the product should be able to carry out for a user.
- Performance Requirements - Specification of criteria that is used to judge the operation of the system from a user perspective. For example, how long it takes a user to perform an operation while utilizing one or multiple functions of the product.
- Environmental Requirements - Also referred to as environmental constraints, these are any restrictions on the product related to which hardware or software to use defined by the client. This will be detailed in Section 4.3.

For the web application, functional and performance requirements, as well as environmental constraints have been identified through the process of requirements acquisition. This process entailed weekly meetings with the client and as a team, which resulted in the culmination of the requirements detailed in Sections 4.2 and 4.3. Fleshing out these requirements will help the team implement this product in the future. Before this, user-level requirements defined by the client will be detailed below.

4.1 User-Level Requirements

User-level requirements are the requirements obtained from the requirements acquisition process with the client. The requirements specifically acquired from the continuous meetings with the are:

- Alleviate the task of a user needing to keep track of a large amount of emails from various clients associated with different projects.
- The product will allow for user accounts to enable secure access courses and projects for students and faculty, as well as any other assisting members of a project, such as a project mentor.
- A faculty member should be able to create courses and projects for students, mentors, and other faculty members to be assigned to.
- Courses and projects should have management utilities that allow for the editing of project details throughout the term of a course. For example, a faculty member may need to remove a student from a project, or they may need to edit an attribute of the project, such as a team's information page.

The above requirements can be refined to represent a clearer understanding of the requirements the product should fulfill. The next section will detail the functional and performance requirements for the product.

4.2 Functional and Performance Requirements

The functional requirements for this project can be divided into several high-level modules. These modules will represent the various stages our web application will need to complete in order to accomplish the desired functionality our client has dictated. These modules follow very closely to our client's workflow detailed earlier in Section 2 of this document. The performance requirements will be split up into the different modules as well and relate to the functional requirements within that module. The separated functional requirement modules are detailed below with their associated performance requirements.

A. **Course Initialization** - This module will include our client's process in reaching out to clients from a client list. The goal is to alleviate our client's task of keeping track of a mass amount of emails within one email inbox, where related emails are not typically in the same thread.
B. **User Account Creation** - Both faculty and students will be able to create an account that will give them access to their assigned courses and carry out their respective necessary tasks for each course.
C. **Team Assignments** - An entire module will be dedicated to outlining the functionality of the process of students being assigned to teams based on their project preferences.
D. **Ongoing Course Management** - After students have been assigned to their teams, faculty may need to edit projects, allow for students to upload deliverables, and create individual team pages.

We will explore and detail the functional requirements of these four high-level modules and their associated performance requirements below. Example use cases are also provided before each module to give a rough idea of what each module is set out to accomplish. For Module A specifically, an example work flow is provided as it supplements the explanation of the module further. Another detail regarding each module is that modules can have similar functionalities between one another. As a result, some modules will be referenced before those modules are detailed, to prevent explaining certain details more than once.

Module A: Course Initialization

Our web application is focused on managing team-based courses in an efficient manner. Below, a flow diagram of our client's current work flow for this module is displayed.

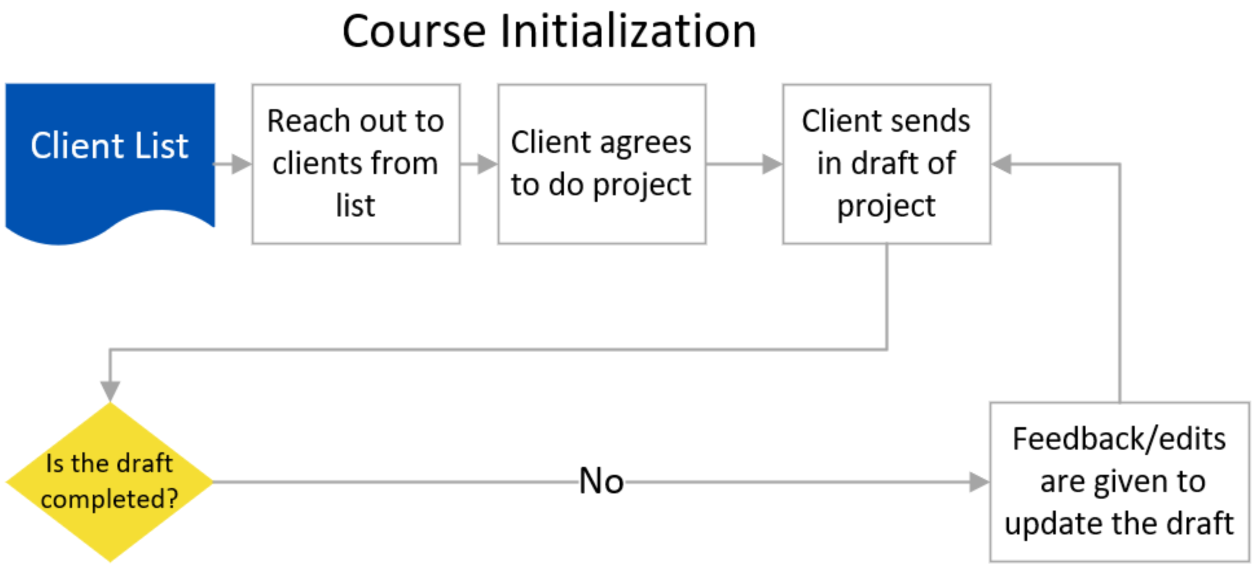
Course Initialization
Client List
Reach out to clients from list
Client agrees to do project
Client sends in draft of project
Is the draft completed?
No
Feedback/edits are given to update the draft

Figure Module A: Diagram showing Dr. Doerry's envisioned workflow utilizing TeamBandit to communicate with clients.

Our client receives constant emails from different clients in different email threads. Keeping track of all of these is an arduous process for our client. In order to fix this current process, email chains will be created for faculty to see all of the emails organized by clients in one place on the web application. The clients and each of their associated emails will also be associated with a project specified by the faculty mentor. In order to show this workflow in action we will show a use case for this module below:

Use Case: Faculty creates an email chain with emails associated with a client email address
User: Faculty Member
Main Flow

1. The system navigates the user to their home page.
2. The user selects the proper course.
3. The system forwards the user to the course page.
4. The user selects a message dashboard icon to take them to their messages related to the course.
5. The system brings the user to the messages dashboard.
6. The faculty member selects the option to add a new client to message.
7. The system displays a form to fill out information about a client.
8. The faculty member fills out information and submits it.
9. The system processes the information and displays a new chat menu to be accessed.

10. The faculty clicks on the new chat menu.
11. The system displays the chat menu regarding the new client.
12. The faculty member copies a unique ID referencing that specific chat menu.
13. The faculty member CC's TeamBandit's email to the email chain associating the client and replies with the unique ID.
14. The system identifies that email chain is associated with the chat menu and pulls information from the email chain to the chat menu.

Alternative Flow

- At any point, if the user decided to back out of the operation, the system will send them back to their previous location on the web page.

In order to help support use cases such as the one above, we specify the functional and performance requirements for course initialization below.

Functional Requirements

A.1 A faculty member can create a faculty account (Detailed more in Module B).

A.1.1 Courses can be created within a faculty account.

A.1.1.1 Faculty can navigate to one of their courses from their TeamBandit home page.

A.1.1.2 Other faculty, as well as mentors and students, can all be invited to a course (Module B).

A.2 A dashboard page for faculty where they can view projects within their courses. This dashboard is essentially a quick snapshot of all teams within a course.

A.2.1 Faculty can create new projects within their course, which can then be viewed in the dashboard.

A.2.1.1 Faculty can add project titles, project descriptions, any necessary logos, clients, mentors, and students.

A.2.2 Faculty can update a status tracker for each project directly on the dashboard. This allows faculty to keep track of where a project stands in terms of their development stages.

A.2.2.1 Within the status tracker, a completion status can be edited by faculty to see where the project currently stands.

A.2.2.1.1 Faculty can choose between pre-determined statuses.

A.2.2.1.2 Faculty can create a custom status.

A.2.3 The dashboard includes a chain of emails from the client(s) associated with a project that a faculty member can interact with.

A.2.3.1 Can read emails between the faculty themselves and the client(s) emails.

A.2.3.1.1 Can read a quick snapshot of the recent emails sent and received. Between the last 3 and 5 emails sent and received.

A.2.3.1.1.1 This snapshot can be quickly expanded to view all emails sent between a client since the email chain began. These individual emails will be colored differently to indicate whether the email has been read or not by the faculty member.

A.2.3.2 Can click a reply button and reply to the email sent.

Performance Requirements

1. 80% of faculty can create a new course and set up basic details about the course within 10 minutes after their account has been created after two attempts.
2. 90% of faculty can manually enter basic information about a project in 3 minutes after one attempt.
3. 75% of faculty can upload files with basic information about a project in 1 minute after one attempt.

Module B: User Account Creation

Every individual using TeamBandit as a means to participate in a course will need to sign up for an account. These individuals include faculty members, mentors, and students. Below we list a use case of a faculty member adding a list of students to associate them with the course:

Use Case: Student accounts can be created via uploaded Excel spreadsheet or CSV file (Assumes the faculty member is already logged into the system).
User: Faculty Member
Main Flow

1. The user navigates to the desired course.
2. The system forwards the user to the course page.
3. The user selects the student list page.
4. The system forwards the user to the student list page.
5. The user selects the option to select a file.
6. The user navigates to the desired file.
7. The system parses the information inside the file.
8. The system adds the information to the student list.
9. The system displays the added students in the list.
10. The user can then choose to mass email the students accounts to send them sign up links.

In order to help support use cases like the one above, we specify the functional and performance requirements for user account creation are detailed below.

Functional Requirements

B.1 Any course organizer can create an account.

- B.1.1 The web application will contain different account types. These include: Faculty, Mentor, and Student.
  - B.1.1.1 Faculty accounts will be created from a basic signup page on the product website.
    - B.1.1.1.1 They have the ability to create their account using NAUauth to link their account to an institution, assuming that institution is Northern Arizona University.
    - B.1.1.1.2 Faculty can invite other faculty member accounts to be co-faculty for a course.
      - B.1.1.1.2.1 A unique invite link will be generated and sent to the other faculty member(s).
        - B.1.1.1.2.1.1 If the invited faculty member already has an account, they will be prompted to sign in. Otherwise, they will be prompted to create an account.
  - B.1.1.2 Both Student and Mentor accounts will be created through a faculty member's account.
    - B.1.1.2.1 The faculty member can decide to manually have a unique link sent to a student or mentor.
      - B.1.1.2.1.1 Faculty will send a unique link to a student or mentor through an email. The specific functionality of sending this email to a student is provided by the product.
        - B.1.1.2.1.1.1 From there, the student or mentor can access the link and finish their account sign up.
    - B.1.1.2.2 Student and mentor information that is needed in order to create their account can be uploaded via an Excel spreadsheet or CSV file by faculty.
      - B.1.1.2.2.1 Contents within the file, such as names and email addresses, will be used in an automated process that sends unique links to all users in the uploaded file.
    - B.1.1.2.3 The faculty member can optionally specify institutional authentication for students.
      - B.1.1.2.3.1 Specifically for Northern Arizona University students and mentors, NAUauth will be used.
- B.1.2 At the minimum, a user will need to sign up with a name, email address, and password.
  - B.1.2.1 Passwords will need to contain an uppercase letter, a number, and a special character. This will be notified to the user as they are

typing their password. The user will not be able to complete their account creation unless they follow the above requirements.

B.2 All registered users will have their own account page.

- B.2.1 Ability for users to manage their profiles.
  - B.2.1.1 Users can add profile pictures.
    - B.2.1.1.1 Users can upload a picture from their local system.
  - B.2.1.2 Users can add personal bios.

Performance Requirements

1. 95% of faculty can sign up for a TeamBandit account in 2 minutes after one attempt.
2. 80% of faculty members can upload a delimited file containing student information in 30 seconds after one attempt.
3. 85% of faculty members can manually enter a single student's details in 30 seconds after two attempts.
4. After all student information has been entered, 90% of faculty members can select to send activation links to the students in 10 seconds after one attempt.
5. 95% of students can select their activation link and complete their account sign up in 2 minutes after one attempt.
6. 95% of users can login to their already existing account in 10 seconds after one attempt.
7. 90% of users can edit their profile pictures and account bios in 2 minutes after one attempt.

Module C: Team Assignments

After both the faculty members and students sign up for a course, students will need to be assigned to teams. These teams will be formed to work on the projects collected in the Course Initialization module. An example of how the faculty members will manually add students to teams is shown as a use case below:

Use Case: Faculty manually assigns a student with a project (Assumes user is already logged into the system).
User: Faculty Member
Main Flow

1. The system navigates the user to their home page.
2. The user selects the proper course.
3. The system forwards the user to the course page.
4. The user selects the team member settings icon on the desired project.
5. The system brings the user to the team settings page.
6. The user selects the student(s) to add to the project.
7. The system displays to the user if the operation is successful.

Alternative Flow

- At any point, if the user decided to back out of the operation, the system will send them back to their previous location on the web page.

In order to help support use cases like the one above, we specify the functional and performance requirements for team assignments below. It is worth noting that Module C contains some functionality provided later in Module D. In some scenarios, they are related in functionality. One example of this is the "project preference" deliverable discussed within Module C. Deliverables as a whole are not detailed in Module C, but in Module D, the functionality of deliverables is explored in more detail. This is also a reason why Module C is the shortest module. The module itself is sparse, but is also vital to our client's work flow.

Functional Requirements

C.1 Mechanism for students to enter project preferences based on a rating schema.

- C.1.1 Will initially be set up as a unique deliverable and will be optional for a course to include.
- C.1.2 Can view all projects and their descriptions in one place.
- C.1.3 Can determine preferences to be visible to the faculty member(s).
    - C.1.3.1 Can edit preferences before a due date specified by faculty.

C.2 Faculty can assign students to projects based on the previously mentioned project preferences.

- C.2.1 Ability to view/download CSV files of students and preferences for external processing aimed at forming teams.
    - C.2.1.1 Can manually select a project and add students by selecting all students that should be assigned to the project based on the project preferences of the student.
- C.2.2 Can choose to automatically have the web application decide on which students should be assigned to certain projects based on the project preferences of the student.
    - C.2.2.1 The exact details of this process have not yet been determined and is considered a stretch goal. It will be further discussed in a future implementation phase.

Performance Requirements

1. 90% of faculty will have the ability to associate a single student with a project in 30 seconds after one attempt.
2. 90% of students will be able to view projects and be able to edit their project preferences simultaneously in 1 minute on their first attempt.
3. 80% of faculty will be able to download a file with information about all student project preferences in 1 minute after one attempt.

Module D: Ongoing Course Management

This section includes the requirements for what a faculty member or a student can do after the course is completely set up and ready for the semester. To demonstrate an example of what a faculty member will be able to do, we detail a use case below:

Use Case: Faculty can archive the course through course settings
User: Faculty Member
Main Flow

1. The system navigates the user to their home page.
2. The user selects the proper course.
3. The system forwards the user to the course page.
4. The user selects the settings icon from the course.
5. The system brings the user to the course settings page.
6. The faculty member selects the option to archive the course
7. The system prompts the user to confirm the archival of the course.
8. The faculty member confirms the archive of the course.
9. The system archives the course.

Alternative Flow

- At any point, if the user decided to back out of the operation, the system will send them back to their previous location on the web page.

In order to help support use cases like the one above, we specify the functional and performance requirements for ongoing course management are detailed below.

Functional Requirements

D.1 Faculty can access the settings of a course.
- D.1.1 Can add new faculty and mentors to a course.
- D.1.2 Can view all projects on a main project dashboard page.
  - D.1.2.1 Faculty can select a specific project to be edited.
    - D.1.2.1.1 Can add students to a project (Module C).
    - D.1.2.1.2 Ability to upload PDF project descriptions.
    - D.1.2.1.3 Can add or remove clients from a project.
      - D.1.2.1.3.1 Faculty can access a page with a client's information.
        - D.1.2.1.3.1.1 Faculty can add or remove as many clients as they wish from a course.
- D.1.3 Within the settings of a course, faculty have the ability to archive or delete a course.
  - D.1.3.1 Faculty can access previously archived courses through the web application.

D.2 Faculty can create deliverables for students to view.
  D.2.1 Students have the ability to upload their deliverables on the web application.
    D.2.1.1 Faculty can grade these deliverables and leave small comments directly on a deliverable on the web application.
    D.2.1.2 Students can update their deliverables.
  D.2.2 Faculty can set a due date for a deliverable.

D.3 Faculty can template information, essentially tailoring course information to their liking.
  D.3.1 Can create a table, such as a table of teams, with any number of rows and columns specified by the user.
  D.3.2 Can create individual team pages to be formatted in a way that students can edit. This ensures that each team page has a similar structure.
    D.3.2.1.1 Faculty can access the settings of a team page.
      D.3.2.1.1.1 Can hide a team page.
      D.3.2.1.1.2 Can publicize a team page.
      D.3.2.1.1.3 Can enter a URL for an external course website

D.4 Users can navigate to a course and view information related to that course.
  D.4.1 Within a course, users can view other user profile pages.
    D.4.1.1 Faculty may hide information on student profiles from other users to view.
  D.4.2 Within a course, users will be able to view and select projects.
    D.4.2.1 Faculty can give users access to specific projects and deny access to certain information about projects users are not associated with.
      D.4.2.1.1 In a project's settings, there will be an accessible setting to the client to decide which users can view a project's details. This will also have the option to be automated.
      D.4.2.1.2 Ability for students to edit their own project details with the proper permissions from faculty.
      D.4.2.1.3 Ability for students to enter a URL to an external team webpage for their team.

Performance Requirements

1. 85% of faculty can begin editing a project's details in 2 minutes after two attempts.
2. 90% of faculty can add or remove a client from a project in 1 minute after one attempt.
3. 75% of faculty will be able to create a team page for one project in 3 minutes after two attempts
4. 80% of faculty can upload a PDF for a single project description in 30 seconds after one attempt.

5. 90% of users can navigate to a course in 20 seconds after one attempt.
6. 80% of faculty members can navigate to and edit a single user's permissions within 2 minutes after two attempts.
7. 90% of users can navigate to a project or team within a course in 30 seconds after one attempt.
8. 80% of faculty members can upload a pdf of a deliverable description in 1 minute after two attempts.
9. 95% of students can upload a pdf of their deliverable in 1 minute after two attempts.
10. 85% of faculty can delete a course from their account in 1 minute after one attempt.
11. 85% of faculty can archive a course from their account in 1 minute after one attempt.

4.3 Environmental Constraints

Environmental constraints refer to restrictions on hardware, software libraries, programming languages, other external standards, laws, or constraints of any sort that our client instructed us to conform to. For this product, the client has not given any environmental constraints, therefore, we have none.

5 Potential Risks

Satisfying both the functional and performance requirements will require the use of specific combinations of technologies that may inherently pose a set of potential risks, each with their own severity and likelihood. These risks might be driven by either the presence of a given technology (such as a database), or the configuration of said technology (vulnerabilities). Minimizing the potential impact of these arising risks is done through an individual means of mitigation aimed at addressing any direct negative effects. Below, a simple table is outlined to show an organized layout of the primary potential risks along with their corresponding severity and likelihood. Note that there are other risks that could be added to this section, such as a member leaving the group. We wanted to limit the amount of risks that had a low likelihood, high severity, and a trivial mitigation. We want to focus on the risks where the mitigations were not obvious at a first glance. Hence, we will only be detailing five potential risks.

RISK	SEVERITY	LIKELIHOOD
Server Goes Down	Medium	Low
Competitor Releases Similar Product	Medium	Low
Team Member Temporarily Unable to Contribute	Medium	Medium
Database Corruption	High	Low
TeamBandit Email Not Carbon Copied (CC)	Medium	Medium

Figure 5.0: Table displaying the primary potential risks with their corresponding severity and likelihood.

There are two distinct kinds of risks that are relevant to our client and our system's functionality: System Risks and Human Error-Driven Risks. System risks are driven by the members of our chosen stack, whereas Human Error-Driven risks are posed by user neglect or mistake.

Risk: Server Goes Down (System Error)

<u>Potential Issues</u>

1) The user would be temporarily unable to interact the system
2) No operations would be able to be conducted by the application

The potential severity of this issue is dependent on two elements:

- The point in time that the server goes down
- The length of time that it remains down.

<u>Mitigation</u>

- Proactive: choosing a well-known and reliable host.
- Monitor downtime regularly
- Keep a log to ensure the chosen server and hardware is reliable prior to deployment of the product

Risk: Competitor Releases Similar Product (Not an error)

Potential Issues

- The need for producing and expediting deployment of new and unique features that distinguish TeamBandit from any clones that may arise in the market

Mitigation

We will simply account for this possibility in advance by:

- Drafting potential innovative features to deploy to keep the competitive edge
- Keeping the system's modularity flexible to simplify the process of later implementing these features

Risk: Team Member Unable to Contribute (Possible human error, but not always an error)

Potential Issues

1) Increasing the workload of the remaining team members
2) Causing a barrier in technological expertise of the incapacitated member
3) Requiring time to be set aside to familiarize the remaining members with the emphasis of the lost member

Mitigation

- Cross-training team members across the different layers of the stack
- Carefully documenting each component of the application's development process

This will ensure no task is only capable of being completed by one member. This is essential so that the remainder of the team can carry on in the event of a loss of a team member without needing to set aside substantial amounts of time to familiarize ourselves with a given technology. With this plan of action being carried out, the only negative result of this risk taking fruition is the increase in the workload from that point on as a result of the newly weighted distribution of the project's development.

As there is with any system that stores essential information, there is an imperative dependency on the database holding that information.

Risk: Corruption or Loss of Information Stored in Database (System Error)

Potential Issues

1) Website data could be inaccurately displayed or omitted altogether
2) Stored email logs could be lost, requiring manual re-entry one-by-one
3) Jeopardize the accuracy of the information
4) Potentially result in components which are dependent on this information to perform improperly

Mitigation

- Make regular backups of all the application's data on a consistent schedule to a reliable location. Additionally, data will be stored without any manipulation unless absolutely necessary.

The most detrimental risk is within the category of human error, and this risk involves the functionality of the application. In order for the system to display emails, a separate email linked to TeamBandit must be carbon copied when the email is sent.

Risk: TeamBandit not Carbon Copied (Human Error)

Potential Issues

1) Essential information to TeamBandit's functionality will be left out - it will have to be manually added by a faculty member of a course.

Mitigation:

There is no deliberate mitigation for this; the solution is simply to manually forward the email to the system. The impact it has on the operation is an inconvenience with a simple solution, and therefore does not warrant an extended plan of action to mitigate.

A fortunate element of this project is that the scale is relatively small technologically. There are not a significant number of risks that pose a serious threat to the system, and those that do exist are not exponentially difficult to prepare for and it is not tedious to minimize their chances of occurring. Now that we have determined our risks and what can be done with those risks to mitigate them, we can now discuss our plan for the remainder of the development cycle.

6 Project Plan

Throughout the first portion of the semester, Team Outlaws has been focused on requirements acquisition and planning, as well as creating complete documentation regarding the steps of our process. As we move into a more software development centric phase of this project, we have designed a project plan. In order to create this project plan, we have designated key components that we believe must be finished in order for other pieces to fall into place, such as creation of the database and a rough frontend shell to begin populating with information from our backend. These technologies will act as cornerstones for the rest of the software to build on, hence their early implementation. Using this philosophy of main components first, we have designed the following Gantt chart:

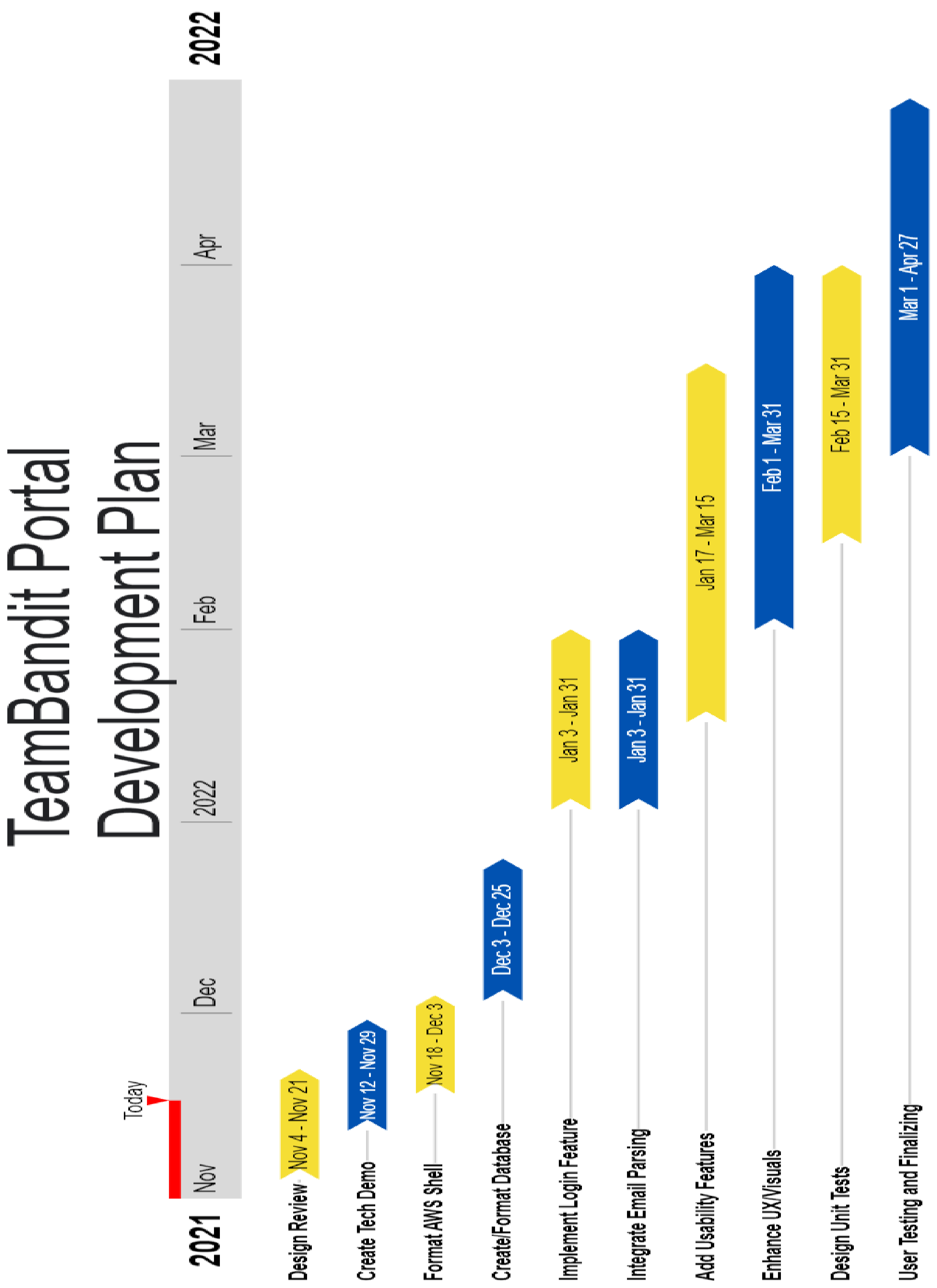

Figure 6.0: This Gantt chart shows the large software development goals moving forward.

This plan follows a mostly linear approach, working from the ground up with necessary features like the database being set up throughout December, with key user features such as login tokens and email integration coming shortly after. Usability features such as custom dashboards, profile pages, and team pages will quickly follow, being built upon the more fundamental components.

Many parts of this development process can be conducted at the same time due to their modularity, including various frontend components. Other portions, however, have to be completed before further work can be done on the application as a whole, such as the creation of the database to store information gathered from the subsequent parts. Towards the end of the semester, the focus will shift from developing new features to iterating upon what has already been developed in order to ensure that the final product is clean and fully functional. By ensuring that we finish the necessities early, we allow more time for our clients and potential users to get hands on with features such as the UI in order to generate feedback for us to act on.

Deliverables will also be a large part of our process next semester, however, these have been left off of the Gantt chart in order to leave room for a more fleshed out development plan.

7 Conclusion

Managing team-based courses is complicated but important. TeamBandit's all-in-one management approach simplifies this by providing a single location where the entire workflow can be carried out. The specific problems in a typical team-based course management workflow are:

- The need to switch between many platforms and applications
- Needing to keep track of many emails from many different people
- The physical need to collect documents from students

The application will allow each of these tasks to be performed via one location on the web where no switching between platforms or hunting down information will be necessary. The specific set of functionality provided will include a panel with options to browse courses, emails and messages, access to a CSV uploader tool, and access documents relevant to the selected team or section. Each option on this panel will be accessible by simply clicking on it, and will redirect the user to the appropriate section of the site.

The processes of this document provided clarity to the details of the

- Specific problem being addressed
- Solution approach and rationale
- Functional requirements

- Performance requirements
- Core use cases of each type of user
- Potential arising risks in the development process

Clarifying the root problem (and hence, solution) that TeamBandit aims to address allows for a more detailed planning process. We first identified the collective subset of problems and difficulties in the typical workflow of managing a team-based course. Then, in direct reference to each of these one-by-one, we outlined two distinct types of requirements: those of what the product's performance should reflect as well as the actual functionalities that it must be capable of carrying out to fulfill its purpose.

The functional requirements of the application were simply identified by translating each step of the typical workflow into a module or set of operations dedicated to optimizing the user's experience conducting that step. This also served as the skeleton for the performance requirements as that section's components are meant to detail the expectations for each of the functional requirements.

The performance requirements detailed the periods of time that each user interaction with the system should take for a majority of users. Each of these aimed to address the most tedious and time-consuming elements of the traditional workflow by minimizing the amount of work and energy required on the user's part.

Setting these requirements and expectations in stone in this document serves as a heavy milestone and contribution to the development process because it can be referenced as a guide to each individual feature and functionality that will need to be set in motion during the implementation phase.

A key insight drawn from these specifications is that there aren't a whole lot of high-stake risks involved in a system of this structure. This means that we will have the ability to put emphasis on the functionality rather than directing heavy focus toward preventing a large index of potential issues.

We are confident that clearly outlining these components provides a plan for establishing each of TeamBandit's objectives and will save time during later phases in the development process. It is expected that ironing out these details now will enable us to focus entirely on the technological aspects of implementation further down the road.