Code Critiquer in C

DESIGN DOCUMENT

Team 34
Client: Iowa State University and Michigan Tech University
Advisor: Dr. Rover

Team Members:

Nicholas Carber	Regex Support
Conner Cook	AST Support
Brandon Ford	Database Admin
Emily Huisinga	Frontend
Sage Matt	Frontend
Cade Robison	Test Suite Support

Team Website:
sdmay24-34.sd.ece.iastate.edu

Team Email:
sdmay24-34@iastate.edu

Revised:
November 30th, 2023
Version 2.0

Table of Contents

List of Figures

List of Tables

Executive Summary

Development Standards & Practices Used

- Software Practices:
  - Code Review: useful to double-check work
  - Software Testing: while annoying, testing can help find mistakes
  - Follow naming conventions for C
- ABET Criteria:
  - Apply knowledge of mathematics, science, and engineering
  - Design a system, component, or process to meet desired needs within realistic constraints
  - Identify, formulate, and solve engineering problems

Summary of Requirements

- The critiquer should be easy and intuitive to use for novice programmers
- The messages from the program should be helpful and descriptive
- The program should catch most of the common antipatterns in C

Applicable Courses from Iowa State University Curriculum

- COM S 185: a basic introduction to C
- COM S 309: understanding project management
- COM S 311: understanding of algorithms
- COM S 317: understanding of software testing
- COM S 327: further development and understanding of C
- CPR E 288: understanding of embedded systems

New Skills/Knowledge Acquired

- Software project management
- Parsing a C program into key parts
- Identifying antipatterns in C
- Crafting error messages that are clear to novice programmers

1 - Team

1.1 Team Members

Nicholas Carber
Conner Cook
Brandon Ford
Emily Huisinga
Sage Matt
Cade Robison

1.2 Required Skill Sets

1. Understanding Of C
2. Knowledge of patterns and anti-patterns in C
3. Understanding of common mistakes from new C programmers
4. Knowledge of CPR E 288
5. Agile Project Management

1.3 Skill Sets Covered

1. Everyone
2. No one
3. Everyone
4. Brandon Ford
5. Everyone

1.4 Project Management Style

Agile structure - sprint length TBD (Probably 2-4 weeks)

1.5 Initial Project Management Roles

Design Document Maintainer / Official Submitter - Emily Huisinga

Website Maintainer - Sage Matt

Main Contact with Dr. Rover - Nicholas Carber

Michigan Tech Liaisons - Brandon Ford, Cade Robinson

System Sketch Maintainer - Nicholas Carber, Conner Cook

Developers - Everyone, to be divided further depending on specifics

2 - Introduction

2.1 Problem Statement

Novice C programmers, specifically CPR E 288 students, need an easy-to-use tool to provide feedback on their C code and help them debug errors in their code. The C compiler alone does not always provide helpful feedback to programmers, especially novice programmers. This critiquer will use compiler feedback and conduct its own analysis of the code to help generate more detailed feedback. The code critiquer will also catch style errors that a normal compiler doesn't check for.

2.2 Requirements & Constraints

UI Requirements

- GUI should be simplistic and easy to use for novice programmers
- All feedback for errors should be presented to the users in a way novice programmers can understand

Maintainability Requirements

- The database should be easy to update with new antipatterns
- The project should be of the same design and standard that Michigan Tech has developed for their other code critiquer in different languages
- The project should be well-documented and easy for another team to pick up when we have completed our part

Functional Requirements

- Files in C should be uploaded successfully
- The program should be able to compile the uploaded code
- Provide proper feedback for at least five compile errors
- Provide proper feedback for at least five style errors
- Allow test suites to be uploaded to run code against to check for runtime errors and check output against expected results
- The program should be able to communicate with a database that stores the C antipatterns
- The program should run through the command line and eventually a GUI

Testing Requirements

- Should be tested with code from both novice and advanced programmers

Legal Requirements

- Have grad students complete CITI training
- Get exempt status from IRB (Institutional Review Board) to use students' code

Performance Requirements

- The code analysis process should take no more than double the time it takes to run the program
- The file upload should take no more than a few seconds and have no loss of data

Constraints

- Time - We have two semesters to get this project designed and implemented
- Storage Space - We may have limited storage for our database where we store data on common C anti-patterns
- Human Ability - We won't be able to think of every possible error or antipattern that a user could run into

2.3 Engineering Standards

First and foremost, we will follow the standards that Michigan Tech has set with their Java and MATLAB Code Critiquers. We will abide by best practices and standards for the C and Python languages as we will use C and Python. Since there is also a strong possibility that we will need to create our own databases, we will ensure our databases and ER diagrams are standardized. The communication between the UI and the backend must also be standardized.

2.4 Intended Users and Uses

This project will benefit any novice programmers working in C, providing them with a user-friendly interface that will debug their code and check for antipatterns. More specifically, this will be implemented in Iowa State's CPR E 288 class to help those students write better C code. Michigan Tech will also eventually implement this alongside the other code critiquers they have written or are working on.

3 - Project Plan

3.1 Task Decomposition

- Create a database to store desired anti-patterns
  - Description: Create a database to store desired anti-patterns and test code.
  - Justification: This is necessary for the functional requirement that the program should be able to communicate with a database that stores the C antipatterns.
  - Subtasks:
    - Create tables with necessary fields in the database
    - Insert anti-pattern data into the tables

- Connect application to database to read from anti-pattern data
  - Description: The frontend application must be connected to the backend database.
  - Justification: This fulfills the functional requirement that the program should be able to communicate with a database that stores the C antipatterns.
  - Subtasks:
    - Set up a toy database to get the initial connection with MySQL
      - This will be a barebones database so that we can test the connection between the application and the database
    - Set up to official anti-pattern database with MySQL
      - This will be the complete database with all of the necessary tables and data

- Handle file upload
  - Description: The application allows users to choose a file to be uploaded to the critiquer system
  - Justification: This fulfills the functional requirement that files in C should be uploaded successfully.
  - Subtasks:
    - Set up a way to prompt the user for a file to upload
    - Read the file data into the application

- System identifies desired compile errors
  - Description: The application can identify at least five compile time errors upon being run with a sample of code.
  - Justification: This is necessary for the functional requirement of catching all compile errors.
  - Subtasks:
    - Groups of similar anti-patterns

- Provide feedback for at least one anti-pattern
    - Description: The application must provide users with useful information on their error and potential ways to fix it.
    - Justification: This fulfills the UI requirement for providing feedback to the user in a way they can understand and the functional requirement for the program should be able to compile and run the uploaded code.
    - Subtasks
        - Terminal feedback
        - GUI feedback

- Test suite support
    - Description: Professors should be able to add tests that can run against students' submitted code to provide custom feedback.
    - Justification: This meets the requirement of allowing professors to test students' code.

- System identifies desired style errors
    - Description: The application can identify at least five style errors.
    - Justification: This is necessary for the functional requirement of catching most style errors.
    - Subtasks
        - Groups of similar anti-patterns

- Allow the professor to add more anti-patterns to the database
    - Description: The professor can add a new anti-pattern to the database through the user interface.
    - Justification: This fulfills the maintainability requirement of the database and should be easy to update with new antipatterns as professors find more.
    - Subtasks
        - Create an interface to submit
        - Connect the interface to the database

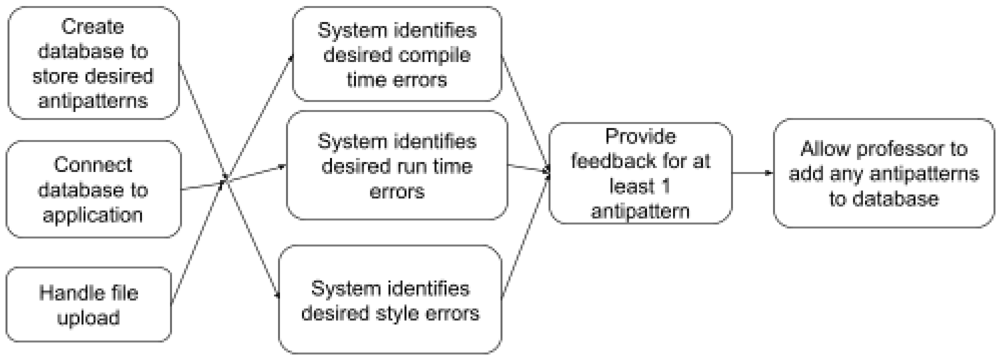
Create database to store desired antipatterns
Connect database to application
Handle file upload
System identifies desired compile time errors
System identifies desired run time errors
System identifies desired style errors
Provide feedback for at least 1 antipattern
Allow professor to add any antipatterns to database

Figure 1 - Task Dependency Chart

3.2 Project Management/Tracking Procedures

We will use the agile project management style as we have all used it before and have found it effective. Dr. Rover and Michigan Tech also have experience with Agile, making collaboration easier.

We will use the GitLab repo and the GitLab issues to track our progress and store our project.

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Key milestones:

- Must recognize at least ten anti-patterns and give valuable feedback
- Should recognize and give feedback on at least five compile-time errors
- Corresponding test suites identify their specified anti-patterns uploaded by professors
- Critiquer does not crash on unrecognized errors

3.4 Project Timeline/Schedule

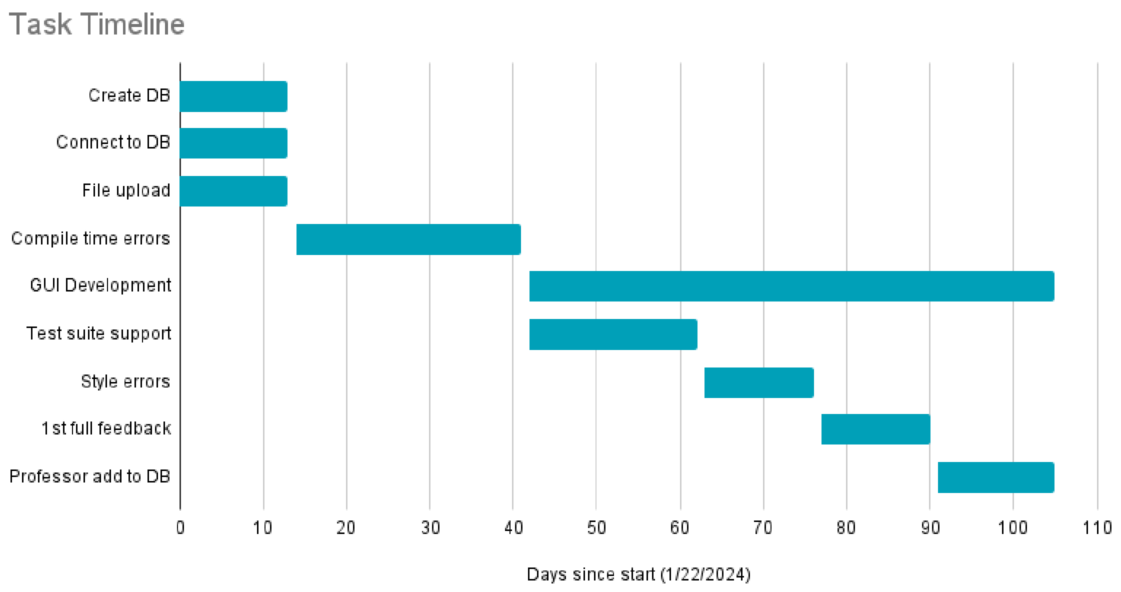

Figure 2 - Gantt Chart/Task Timeline

Task name	**Start date**	**End date**
Create DB	1/22/2024	2/4/2024
Connect to DB	1/22/2024	2/4/2024
File upload	1/22/2024	2/4/2024
Compile time errors	2/5/2024	3/3/2024
GUI development	3/4/2024	5/6/2024
Test suite support	3/4/2024	3/24/2024
Style errors	3/25/2024	4/7/2024
1st full feedback	4/8/2024	4/21/2024
Professor can add to DB	4/22/2024	5/6/2024

Table 1 - Task Start/End Dates

3.5 Risks And Risk Management/Mitigation

Agile projects can associate risks and risk mitigation with each sprint.

Risks:

1. Create Database: database is not created, 0.1
2. Connect To Database: application can't connect to database, 0.1
3. File Upload: file is not uploaded correctly, 0.5
    a. Reasoning: Users could potentially upload malicious files.
    b. Solution: We could run code in its own environment so it will not harm or interfere with our application.
4. Provide feedback for antipatterns: feedback is not printed correctly, 0.1
5. System identifies compiler errors: compiler errors are not identified or identified incorrectly, 0.2
6. System identifies run-time errors: runtime errors are not identified or identified incorrectly 0.3
7. System identifies style errors: style errors are not identified or identified incorrectly 0.3
8. Allow professors to add antipatterns to the database: professors cannot add antipatterns to the database, 0.5
    a. Reasoning: Users could potentially add erroneous or malicious antipatterns to the database.
    b. Solution: Each user has their own local database for storing new antipatterns

3.6 Personnel Effort Requirements

Task	Effort Points	Explanation
Create database for anti-patterns		
Create tables	2	Only need 1 or 2 simple tables
Insert anti-patterns	20	Need to create regular expressions from anti patterns
Connect application to database		
Create toy database	2	Simple initial connection
Connect actual database	3	Experience from initial connection will make this easy
Handle file uploads		
Prompt user for file	2	Command line prompt - "Enter filename:"
Read file data into application	4	Load a file into the application and make sure we can access the data after building an Abstract Syntax Tree
System identifies errors		
Compile errors	30	Need to worry about multiple files. Need to run the program through regex data.
Run-time errors	40	Textual analysis done without running the program
Style errors	15	Match the regexes from the database
Provide feedback for antipatterns		
Terminal feedback	5	Output formatted feedback generated by the program
GUI feedback	15	Have to build a GUI that will eventually connect to the main application
Allow professors to add antipatterns to the database		
Create interface to add antipatterns	5	Basic form for new antipattern information, which then gets stored in the database
Connect interface to database	5	Create an SQL INSERT statement to put the new anti-pattern data into the database

Table 2 - Task Points and Descriptions

3.7 Other Resource Requirement

We have established that we will need some sort of server to host the anti-pattern database. A temporary solution for this could be the High-Performance Clusters we can access through class. For a longer-term solution, we can ask Michigan Tech if they have a server we can use.

4 Design

4.1 Design Content

Our first design concept for the Code Critiquer is as follows. The Code Critiquer will be, first and foremost, an application where students can upload their work with the ability to configure what errors they want to look for. They would receive feedback similar to compiler feedback, except more specific and understandable way by even programming novices. Eventually, the application would be hooked up to Canvas as an External App using LTI tools. However, after further research and understanding of our project, our first design needed to be revised. Please reference section 4.7.2 is our updated design for the Code Critiquer.

4.2 Design Complexity

The design we came up with includes multiple components such as the GUI, main application, abstract syntax tree builder, the database, and the Canvas LTI module. One engineering principle we use is the layered architecture structure. This is the architecture design where each module can only access the layer above and below it. This limits each component's dependencies and lowers the chance of a circular dependency.

One challenging requirement is to generate valuable feedback for errors in C code. Currently, the GNU C compiler doesn't provide feedback on how to fix certain errors. Another challenging requirement is allowing professors to upload tests to run students' code with.

Additional complex components of the critiquer are generating the Abstract Syntax Tree and the corresponding XPath. These are two very important aspects of the Critiquer required to identify a subset of Anti-Patterns easier compared to a Regular Expression.

4.3 Modern Engineering Tools

For the development of this design, we will use draw.io to create a frontend layout. Depending on the developer's preference, we will use either Visual Studio Code or JetBrains software to develop the code.

A big tool we are leveraging is the Clang Compiler and its Static Analyzer. This system allows us to quickly generate the required foundational structures (abstract syntax tree) for identifying antipatterns in the student's code. We will also employ the Python language to develop our system as it provides easy-to-use libraries and syntax. The database will be through MySQL.

4.4 Design Context

Code Critiquer in C is designed for programming learners to provide instant feedback on their code and improve their programming skills. It also benefits teachers, as it can be used in tandem with coursework and save them valuable time that would be spent correcting simple errors.

Area	Effects of Code Critiquer in C
Public health, safety, and welfare	This project will greatly improve the mental health of students struggling to learn programming but feel that they have nowhere to go for help to learn.
Global, cultural, and social	There are several ways that Code Critiquer in C can both positively and negatively impact our profession. It can be used to create a generation of programmers who have incredibly strong programming skills. On the other hand, if the Code Critiquer in C were to become good enough, future programmers may become lazy and rely on the critiquer to make corrections. It would be similar to how our generation has become increasingly poor spellers as we rely on autocorrect.
Environmental	This project will have a minimal environmental impact, except for the energy it takes to run the device on which the application will be run.
Economic	This project has the potential to inflate the market with programmers, as students will feel more supported during the learning process and, therefore, will be less likely to drop out. Additionally, if the Code Critiquer in C technology could become advanced enough, it could decrease the demand for programming tutors.

Table 3 - Effects of Code Critiquer in C

4.5 Prior Work/Solutions

We are extremely fortunate to have contact with Michigan Tech, who have already created Code Critiquers in MATLAB, Python, and Java. They have graciously allowed us to analyze their code critiquers to aid in our own development.

Michigan Tech's Python Code Critiquer:

Advantages: Due to the many useful libraries that Python has available, we have decided to create our C Code Critiquer in Python. As such, the Python Code Critiquer, written in Python, will have the most similar code structure to ours.

Shortcomings: As C is a much lower-level language and has the tricky issue of memory management, our C Code Critiquer will become much more complex than the Python Code Critiquer.

Michigan Tech's Java Code Critiquer:

Advantages: Michigan Tech's Java Code Critiquer has the most thorough database of antipatterns and will be most useful to us, as Java and C have their similarities (though many, many differences).

Shortcomings: The Java Code Critiquer is written in Java, and as we will be writing our Code Critiquer in Python, this code structure will be useless to us.

Michigan Tech's MATLAB Code Critiquer:

The MATLAB Code Critiquer offers nothing unique to us that the Python and Java Code Critiquer don't already offer. Nevertheless, it is still a useful tool in our pockets.

Pros for our Code Critiquer In C compared to other Code Critiquers:

As far as we are aware, there has been no Code Critiquer made for a language as low level as C. Thus, our project will fill an important gap in autonomous code critiquing for students learning C.

Cons for our Code Critiquer In C compared to other Code Critiquers:

N/A

4.6 Design Decisions

1. We had to decide to allow professors to upload their own code.

2. We had to decide to implement it as a command-line application before creating a GUI/importing it to Canvas

3. We had to make decisions to allow for personal configurations. For example, to choose whether you want to check for certain types of errors, or whether you want to critique only one file or multiple files via a Makefile.

4.7 Proposed Design

4.7.1 Design v1 (Initial Design)

The C code critiquer system consists of 3 main components: the website, the database, and the critiquer server. After the student logs in, they can upload their code to the website or canvas. From there, the code critiquer logic will be called from the server. This component will check the uploaded code against the antipatterns stored in the database. After finding all the errors in the student’s code, the critiquer will return the feedback to the student through where the code was uploaded.

System Sketch

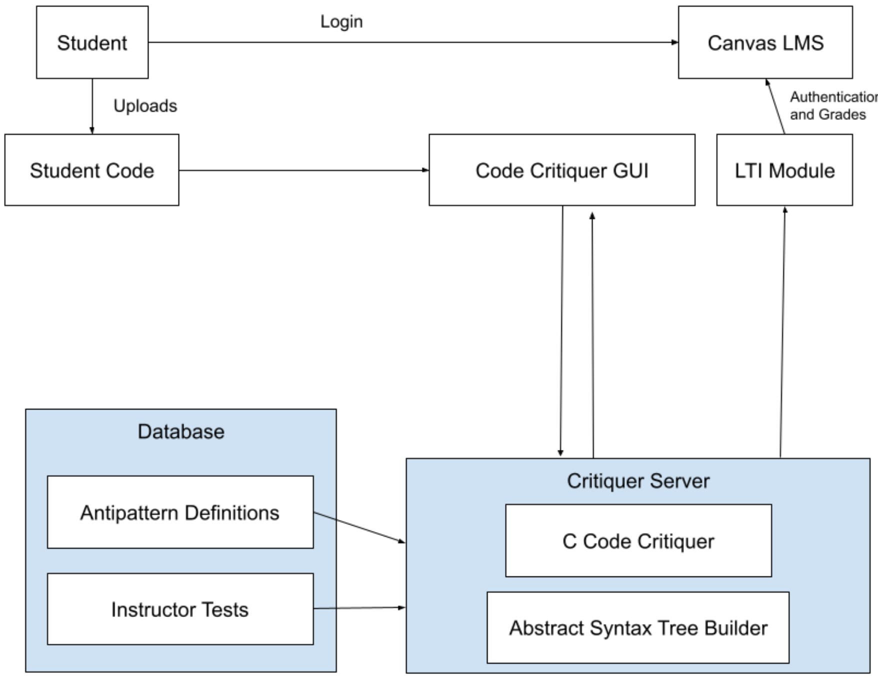

Figure 3 - System Sketch v1

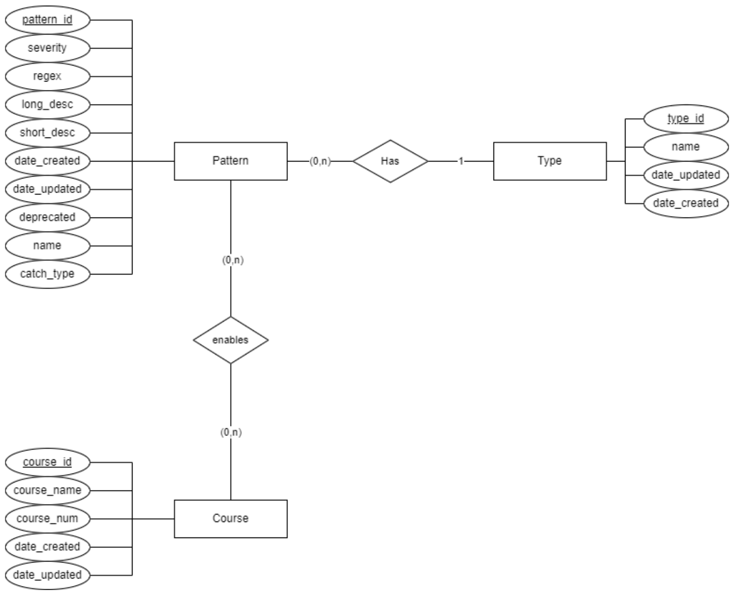

Figure 4 - ER Diagram v1

Student Code: The student will submit the code files. These can be in the form of a single C file or a makefile for compiling multiple files. This should fulfill the functional requirements of allowing a student to upload their code and the performance requirements of uploading the code in an acceptable amount of time. This is a vital aspect of the functional requirement that students should be able to upload code to the application.

Code Critiquer Application: This interface will allow students to interact with the system (upload code, see results, etc). This will help fulfill many of the functional requirements, such as students being able to upload code and receive feedback on errors.

LTI Module: Connection point between Canvas and the Code Critiquer System. The LTI Module follows standard practice for connecting LMS and LTI systems. This will help fulfill the UI Requirement that the application will have to be run through a GUI (in this case, Canvas).

Canvas LMS: This is the Canvas everyone knows and loves here at Iowa State. Eventually, using the LTI Module, we will connect the application to Canvas for students to upload their code and receive feedback. This will fulfill the UI Requirement that the application will be run through a GUI (in this case, Canvas).

Critiquer Server: This will consist of the code critiquer application and the abstract syntax tree builder. The code critiquer will analyze the student's code using the anti-pattern data in the database and generate appropriate feedback based on any errors found. The abstract syntax tree builder reads in the students' code and generates an abstract syntax tree for the critiquer to analyze. This will help fulfill the functional requirement that students will receive feedback from antipatterns stored on a database (which will be hosted on the server).

Database: The database will hold all the information for the antipatterns to check against the uploaded code. This is critical to the software's functionality as the system needs a place to store all the antipatterns. The database will also hold the tests the instructors upload for the particular assignment. This is also crucial to the requirement that the system needs to run the uploaded code. Without these tests, there is no way to run the code and have an expected output. This will help fulfill the functional requirement that students receive feedback from antipatterns stored in a database.

Functionality

The Code Critiquer will be, first and foremost, an application where students can upload their work. They can either upload a single file to get it critiqued or upload a project, which would require a makefile. Once the file or project is uploaded, they would click a very prominent "Critique" button and receive feedback. On the backend, the application would be hooked up to a database with antipatterns, and the program would scan the file or project for the presence of these antipatterns. The feedback would be similar to compiler feedback, except more specific and worded understandably by even programming novices. Eventually, the application would be hooked up to Canvas as an External App using LTI tools.

The current design satisfies the functional and non-functional requirements very well.

4.7.2 Design v2

Updated design from 4.7.2.

System Sketch

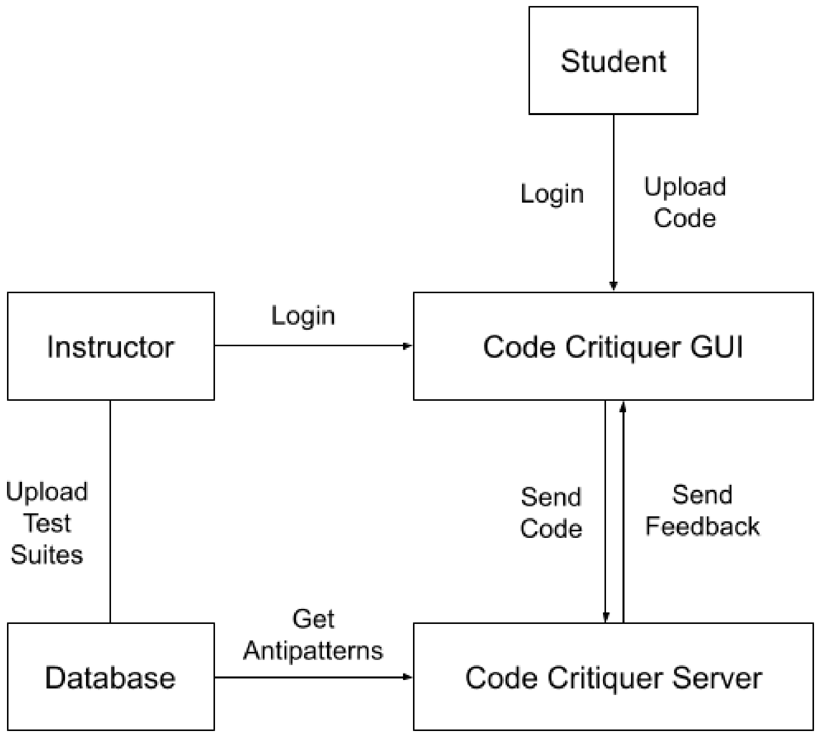

Figure 5 - System Sketch v2

ER Diagram of the Database

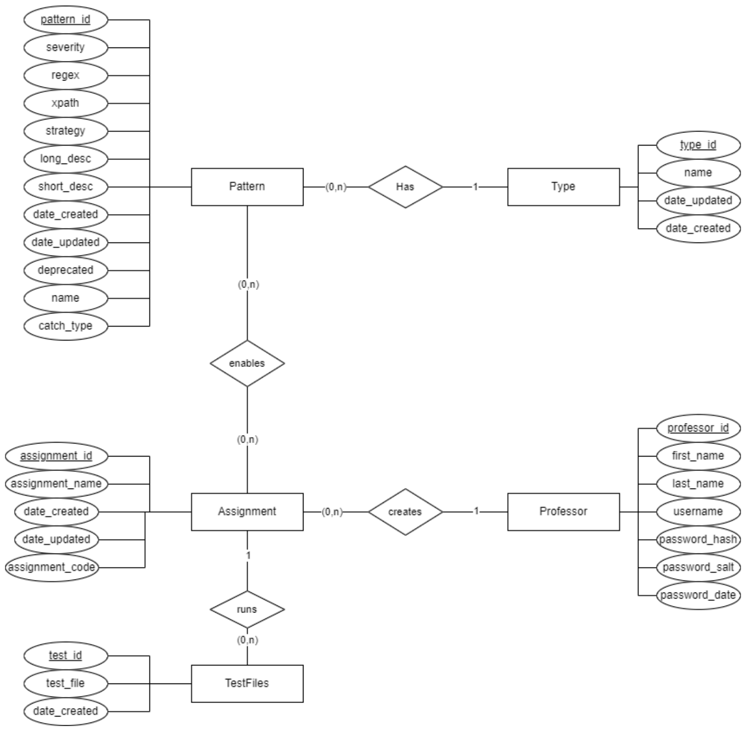

Figure 6 - ER Diagram v2

Component Diagram

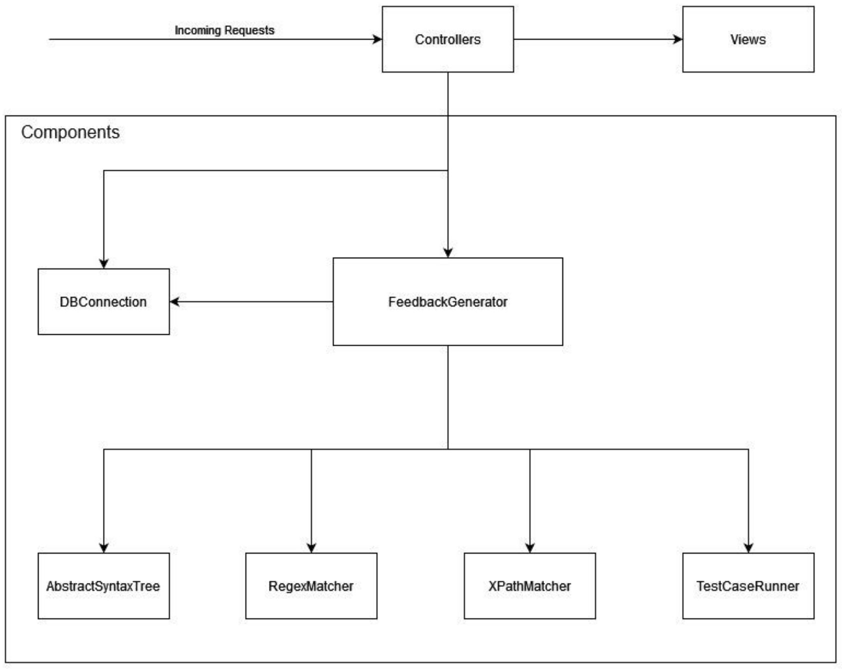

Figure 7 - Component Diagram

UI Diagram

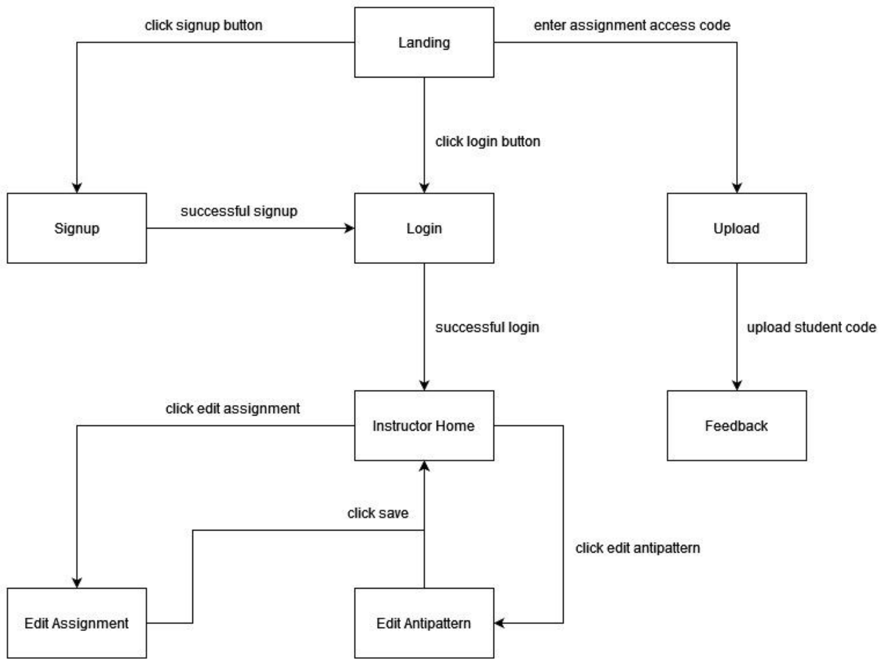

Figure 8 - UI Diagram

Functionality

The second iteration of the design is similar to the first in most ways. The only change was removing Canvas integration due to the difficulty of implementing this in the given timeframe.

4.8 Technology Considerations

One huge technology consideration that had to be made was what language to write the C Critiquer in. So far, Michigan Tech has followed the pattern of creating their critiquers in the same language that their critiquers are critiquing. That would mean that, if we followed this pattern, we would create our critiquer in C. However, as a relatively low-level language, C needs more libraries than higher-level languages have. As such, we have decided to break this pattern and write the C Critiquer in Python. Python already has many libraries useful for testing C code. Utilizing these libraries will help us use our time best and create the best final application.

Another consideration is what to use for a server to host the database. Short-term, we plan on using high performance clusters offered through this class. Long-term, we plan to use the servers that Michigan Tech already has.

We have decided to use MySQL for our database as we all have experience with it from other classes. Finally, for our IDEs, we have decided to use both PyCharm and Visual Studio Code, depending on the individual programmer.

4.9 Design Analysis

Initially, our design in 4.7.1 included plans to integrate the Code Critiquer with Canvas as an external application with LTI Tools. However, Michigan Tech received feedback from their students that they did not like the Canvas interface. As such, we decided to remove the plans to integrate with Canvas and instead have the Code Critiquer as an independent application. Additionally, we have decided to move from using React to using Flask instead as a framework. This will allow us to centralize our code base with Python. Otherwise, as we have been working on the project, identifying antipatterns, and preparing feedback for the antipatterns, our initial design has held up well.

There are many opportunities for improvement for the C Critiquer. We can always add the ability to catch more antipatterns. Additionally, we can add more opportunities for teachers and students alike to customize the C Critiquer to their preferences. This includes configuring what antipatterns are identified, adding multiple classes with unique configurations, and adding multi-file projects.

5 Testing

Our testing strategy uses the unittest library to test our Python code which makes up most of our code base. For any JavaScript code we include in our user interface, we will use Jest. One of the challenges we will face when doing our acceptance testing is that we will need to use students' code to test that the system generates the correct feedback and is usable from a student's perspective.

5.1 Unit Testing

The framework we will be using to test our software will be the unittest library for Python. This should help us easily test all the functions needed for each class. Additionally, we will use the Jest framework when testing individual units on our UI that use JavaScript. We can ensure each component is working by breaking down our system into these units.

5.2 Interface Testing

Some of the interfaces in our design will be the Abstract Syntax Tree, the regex matcher, the User Interface, and the feedback generator. Because our code will be almost entirely Python, we will use the unittest library to create test suites for those interfaces. For any JavaScript we include in our user interface, we will use Jest to test that code.

5.3 Integration Testing

We will have to test three main communications for integration. The first is the communication between the application logic and the database. We will test this by attempting to connect to the database and checking for connection errors. We will be using the MySQLdb library for connecting the application to the database so we will have to test any functions we create using this library.

The second integration test category is UI communication. We must test that the UI can effectively transmit data from the user to the application logic. Again, we will have to ensure no errors occur when connecting to the application and that no data is corrupted when making the connection. This will be done by testing manually.

5.4 System Testing

We will provide the system with example code and test that the system generates the expected results. The feedback should be based on the data we have stored in the database, and it should be displayed correctly. This will be done using several simple pre-generated code blocks, supplying them to the critiquer system, and comparing the generated output with our pre-defined expected results. By doing this, we can verify, given the same code, that the system works as expected, yielding consistently correct responses.

5.5 Regression Testing

Regression testing will help ensure that new code pushed to GitLab won't break the current functionality. Any previously written tests can help catch unexpected changes. Therefore, all the tests described in previous sections will be helpful with regression testing.

Writing tests for every component, even when it seems straightforward and unlikely to break, can help catch changes that break unexpected parts of the code. Keeping all tests can also help catch unexpected effects on old code. When writing code locally, ensuring that our local codebase is up to date (pulling from GitLab frequently) can help avoid merge conflicts. Running all tests before pushing code to GitLab can ensure breakages are found before being merged into the codebase. Running tests with a local copy of the database can also ensure changes don't break anything in the global database.

5.6 Acceptance Testing

- Functional
  - Goals for compile time, runtime, and style errors should be met with appropriate feedback. Specifically, successfully give feedback on at least five errors in each category.
  - Critiquer should not extend the time to run a program by more than double. (estimate - can change)
- Non-Functional
  - Command-line and GUI should be intuitive for novice programmers.
  - Modular database with easy extensibility.

Functional tests will have either unit tests or performance tests that pass for each requirement. Non-functional tests will be reviewed by our faculty mentor, Michigan Tech's team, and some beginner programmers from Iowa State.

5.7 Security Testing

We will be storing user data in our database, so we will need to be able to encrypt sensitive data that will be stored. We will use predefined and proven encryption methods, so we will not need to test said functions. However, we must ensure data cannot be leaked from our database. Therefore, we should sanitize all of our SQL queries. Also, since we are allowing professors to upload their own code to test the student's code, we need to make sure that malicious code cannot alter the application's behavior in any way. Our solution for this is to run the professor's code in its own environment so that it can't impact the execution of the application. Testing these functionalities should be pretty straightforward, as we just need to make sure all queries are sanitized and no plaintext of sensitive information makes it into the database.

Furthermore, to prevent injection attacks through uploaded code, we will test to ensure that the code uploaded will run in its own environment and, in the event of a crash of said environment or a breach of said environment, the application environment will not be affected.

5.8 Results

By developing tests to cover each aspect—unit, integration, system, and so on—we can ensure that we meet functional requirements. By respecting the testing process, we ensure that we follow best practices to develop a system that meets the guidelines we have previously identified. As a whole, testing provides both verification and validation of our system.

6 Implementation

We have planned out the project structure/architecture and come up with a list of all the code files we will need. The model-view-controller (MVC) design pattern is how we designed our project structure. The models would be our logic components that critique the student's code and will be written in Python. The views are the frontend files that deal with the website user interface. This includes any HTML, CSS, and JavaScript files. The controllers will handle the request mapping, run the logic components as needed, and return the correct views for the page the user is trying to access. The controllers will be written in Python using the Flask microframework.

Models:

- DBConnection
  - Handles any interaction with the database
- FeedbackGenerator
  - Starting point for kicking off the critiquer
  - Runs AST, regex, xpath, and test cases
  - Returns feedback data as JSON
- AbstractSyntaxTree
  - Creates AST from student code
  - Detects antipatterns that are found with the AST
- RegexMatcher
  - Selects all antipatterns from the database that have a regex expression stored
  - Uses the regex to detect those antipatterns
- XPathMatcher
  - Selects all antipatterns from the database that have an xpath expression stored
  - Uses the xpath to detect those antipatterns
- TestCaseRunner
  - Runs any test code that the professor has uploaded for the assignment

Views:

- Landing
  - Enter code for students, log in or sign up as instructor
- Signup
  - Takes in information to create an account
  - Redirects to log in on account creation
- Login
  - Takes email and password to login

- Instructor Home
    - Home page for instructors
    - Shows assignments and custom antipatterns
    - Can edit an assignment or antipattern or create a new one
- Edit Assignment
    - Form for all fields related to storing an assignment
    - Can select / deselect custom antipatterns to be used for this assignment
    - Upload test cases here
- Edit Antipattern
    - Form for all fields related to storing an antipattern
- Code Upload
    - Page where student code is uploaded
- Code Feedback
    - Displays code feedback

Controllers:

- __init__.py
    - Import necessary libraries for Flask
    - Start Flask application
    - Register any error handling for bad requests, etc
- LandingController
    - '/' GET mapping
        - Return landing.html
- SignupController
    - '/signup' GET mapping
        - Return signup.html
    - '/signup' POST mapping
        - Handle account creation
        - Redirect to '/login' on successful account creation
- LoginController
    - '/login' GET mapping
        - Return login.html
    - '/login' POST mapping
        - Handle login, create session for user
        - Redirect to '/instructor/home on successful login
- InstructorHomeController
    - '/instructor/home' GET mapping
        - Fetch instructor data using session info
        - Return instructor_home.html
- EditAssignmentController
    - '/edit/assignment/{id}' GET mapping
        - Return edit_assignment.html with current values filled in

- ‘/edit/assignment/{id}’ PUT mapping
  - Update assignment details in database
  - Redirect to ‘/instructor/home’
- CreateAssignmentController
  - ‘/create/assignment’ GET mapping
    - Return edit_assignment.html with blank values filled in
  - ‘/create/assignment’ POST mapping
    - Create new assignment in database
    - Redirect to ‘/instructor/home’
- EditAntipatternController
  - ‘/edit/antipattern/{id}’ GET mapping
    - Return edit_antipattern.html with current values filled in
  - ‘/edit/antipattern/{id}’ PUT mapping
    - Update antipattern details in database
    - Redirect to ‘/instructor/home’
- CreateAntipatternController
  - ‘/create/antipattern’ GET mapping
    - Return edit_antipattern.html with blank values filled in
  - ‘/create/antipattern’ POST mapping
    - Create new antipattern in database
    - Redirect to ‘/instructor/home’
- CodeUploadController
  - ‘/code/{access_code}/upload’ GET mapping
    - Return code_upload.html
  - ‘/code/{access_code}/upload’ POST mapping
    - Upload file to server and start analysis
    - Show loading screen until finished then redirect to ‘/code/{access_code}/feedback’
- CodeFeedbackController
  - ‘/code/{access_code}/feedback’ GET mapping
    - Return code_feedback.html

7 Professionalism

This discussion concerns the paper titled "Contextualizing Professionalism in Capstone Projects Using the IDEALS Professional Responsibility Assessment," International Journal of Engineering Education Vol. 28, No. 2, pp. 416–424, 2012

7.1 Areas of Responsibility

Area of Responsibility	Definition	NSPE Canon	How SE code addresses responsibility	Difference between SE code and NSPE
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence.	Perform services only in areas of their competence; Avoid deceptive acts.	SE 1.03 specifies that one should be qualified for the work that they are doing. SE 6.08 says to avoid making deceptive claims about your software.	SE 6.08 says to avoid making deceptive claims about your software, but nowhere does it say to avoid deceptive acts in general.
Financial Responsibility	Deliver products and services of realizable value and at reasonable costs.	Act for each employer or client as faithful agents or trustees.	SE 1.14 addresses this as it aims to promote maximum quality with minimum cost to employer, client, user, and public.	The SE code addresses the monetary and quality of the final product, whereas the NSPE specifies that one should act faithfully to their employer or client.
Communication Honesty	Report work truthfully, without deception, and understandably to stakeholders.	Issue public statements only in an objective and truthful manner; Avoid deceptive acts.	SE 2.06 specifies that statements should be truthful and fair, especially regarding software and related documents.	SE emphasizes that SE documents must be faithful regardless of public or not whereas NSPE is more overall on all public statements.
Property Ownership	Respect the property, ideas, and information of clients and others.	Act for each employer or client as faithful agents or trustees.	SE 4.03 and 7.03 discuss only using the property of others in proper and authorized ways and giving full credit to others.	NSPE focuses on the relationship of acting in good faith with the employer or client, whereas SE is giving credit to all regardless of whom they are addressing.
Health, Safety, Well-Being	Minimize risks to the safety, health, and well-being of stakeholders	Hold paramount the safety, health, and welfare of the public.	SE 6.10 states the need to obey all laws governing their work to be consistent with public health, safety, and welfare.	SE references governing rules and laws to prioritize public safety, whereas the NSPE is more general, keeping it at the forefront.

Sustainability	Protect the environment and natural resources locally and globally.		SE 2.02 mentions that software should be approved only if it meets appropriate tests and does not diminish the quality of life or harm the environment.	SE code addresses the responsibility of sustainability, whereas the NSPE table does not mention it.
Social Responsibility	Produce products and services that benefit society and communities.	Conduct themselves honorably, responsibly, ethically, and lawfully to enhance the honor, reputation, and usefulness.	SE 6.02 says to ensure that people know about the SE code of ethics and Software Engineer's code of ethics and their responsibility to it.	Although the SE Code often indirectly discusses honor, ethics, and responsibility, it does not say to do these things specifically to enhance the profession.

Table 4 - Society-Specific Code of Ethics

7.2 Project-Specific Professional Responsibility Areas

Financial Responsibility does not apply to our team, as we will not be spending money on this project. The only cost associated with this project would have been the servers, but as we are utilizing Michigan Tech's servers, this cost is nonexistent. The performance level is N/A.

Communication Honesty is an area that is very applicable to us. We must communicate clearly to both users and stakeholders so that it is clear what is happening with the code and other data. We are performing very well in this area, so the performance level is high.

Property Ownership is another important area. We must ensure that the code's safety in the database is never compromised, as that code is their ideas and properties and must be protected as such. As of right now, we are performing high in this area.

Health, Safety, and Well-being is an area that can be of concern to us. Regarding safety, we must ensure that any sensitive data about our users is never compromised. Regarding mental health, our application will offer many improvements to students everywhere. Otherwise, in terms of physical health, our application will have little ef on physical healthfect. We are currently performing high in this area.

Sustainability is another area that does not apply to our team. The only environmental impact is the energy it takes to run the application, which is extremely minimal. So, this performance level is N/A.

Social Responsibility is another important area. Our application, once finished, will be a great benefit to society. We must ensure that this does not become a tool for malevolence. As of right now, we are performing high in this area.

7.3 Most Applicable Professional Responsibility Area

Work Competence is the most applicable professional responsibility area. Our team needs to deliver a high-quality end application that correctly identifies all the errors promptly. Currently, we have a medium performance level in this area. There is room for improvement in terms of timeliness.

8 Closing Material

8.1 Discussion

We now have a solid foundation for the implementation of the Code Critiquer. We have documented many antipatterns for later implementation and have an understanding of what each of our components is going to look like. Additionally, we have a solid plan to build on moving forward in the semester. All our requirements will be met by the deadline if we follow the plan outlined in this document.

8.2 Conclusion

Our main goal has been to fully research and design our system to ensure smooth implementation. We have met this goal by clearly defining our components and how they will interact. Additionally, we have identified a number of antipatterns for the Code Critiquer and have been constructing helpful feedback for each antipattern. We have also planned how to test all code developed for this project to ensure that the project can continue development after our year is done.

8.3 References

[1] Ureel, L. C., Brown, L., Sticklen, J., Jarvie-Eggart, M. E., and Benjamin, M. , "Work in Progress: The RICA Project: Rich, Immediate Critique of Antipatterns in Student Code," in *Proceedings of the 6th Educational Data Mining in Computer Science Education (CSEDM)*, 2022, 75-81. http://doi.org/10.5281/zenodo.6983498

[2] Ureel, L. C., and Wallace, C. , "Automated Critique of Early Programming Antipatterns," in *SIGCSE '19: Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2022, 738-744. http://doi.org/10.1145/3287324.3287463

8.4 Team Contract

Team Members

- Nicholas Carber
- Conner Cook
- Brandon Ford
- Emily Huisinga
- Sage Matt
- Cade Robison

Team Procedures

1. Day, time, and location for regular team meetings
    - Thursday 1:00-1:45 hybrid (Discord or library), Sundays case-by-case
2. Preferred method of communication updates, reminders, issues, and scheduling
    - Discord
3. Decision-making policy:
    - Majority vote
4. Procedures for record keeping:
    - Shared document that everyone can contribute to

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:
    - Be on time and communicate if you can't attend
2. Expected level of responsibility for team assignments, timelines, and deadlines:
    - Attempt to complete all tasks on time to the best of your ability
    - Reach out for help if needed
3. Expected level of communication with other team members:
    - Reach out to team members if you are struggling with a task in the project
    - Keep team members updated on progress
    - Acknowledge messages in Discord
4. Expected level of commitment to team decisions and tasks:
    - Everyone should contribute to conversations
    - Hold each other accountable

Leadership

1. Leadership roles for each team member:

- Nicholas Carber - Regex Support
- Conner Cook - AST Support
- Brandon Ford - Database Administrator
- Emily Huisinga - Frontend
- Sage Matt - Frontend
- Cade Robison - Test Suite Support

2. Strategies for supporting and guiding the work of all team members:

- Use GitLab's issue board to weight issues based on estimated workload
- Distribute the issues to team members evenly, with consideration for an individual's pre-existing commitments and overall availability to work.

3. Strategies for recognizing the contributions of all team members:

- Issues will be assigned, so there will be a record of work done

Collaboration and Inclusion

1. Skills, expertise, and unique perspectives each member brings to the team:

- **Brandon:** I have taken CPR E 288, so I have had some experience with embedded systems topics and programming. I have also taken SE 185 and CPR E 308, which have given me more C programming experience. In addition to this, I work as a computer science tutor, so I have some idea of the common issues novice programmers run into.
- **Sage:** I've had various internships and working experiences, which include working with WPF application development, MATLAB, and most currently embedded systems and C.
- **Cade:** I have had experience with agile development as a part of a scrum team during an internship. I have also had a lot of experience programming in multiple languages for classes here at Iowa State. I also have experience teaching inexperienced Computer and Software engineers as a TA for CPRE 281.
- **Nicholas:** I had an internship working in an agile development cycle. I've also done some data science work. I've also written a lot of tests: unit, functional, and performance.
- **Emily:** I've had much experience with an intense code review process (my internship required 2 code reviews and one quality assurance review). I also was a TA for SE/CPRE 185 for five semesters, so I have experience working with novice programmers.
- **Conner:** I have a psych and biomedical minor, so maybe that could offer some new perspective.

2. Strategies for encouraging and supporting contributions and ideas from all members:

- Ensure that everyone is being listened to and respected.

3. Procedures for identifying and resolving collaboration or inclusion issues:

- First, try to work it out inside the group. If you are uncomfortable speaking with the person individually, reach out to another group member to try and open a discussion. If escalation is needed, first reach out to the TA, then the professor.

Goal-Setting, Planning, and Execution

1. Team goals for this semester:

- Have fun
- Make a functional application
- Keep on time with meetings
- Complete required tasks by the end of each sprint

2. Strategies for planning and assigning individual and team work:

- Maintain issue board
- Assign tasks as equally as possible

3. Strategies for keeping on task:

- Set strict time slots for meetings
- Set strict deadlines for tasks

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?

- Bring them up at team meetings if necessary; otherwise address them on Discord.

2. What will your team do if the infractions continue?

- Address with TA or professor if necessary.

****

a) I participated in formulating the standards, roles, and procedures as stated in this contract.

b) I understand that I am obligated to abide by these terms and conditions.

c) I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.

1) Sage Matt	DATE: 12/3/23
2) Brandon Ford	DATE: 12/3/23
3) Conner Cook	DATE: 12/3/23
4) Emily Huisinga	DATE: 12/3/23
5) Nicholas Carber	DATE: 12/3/23
6) Cade Robison	DATE: 12/3/23