Code Critiquer System for the C Language and Embedded C

Design Document

Team sdmay25-23
Client: Dr. Diane Rover
Advisor: Dr. Diane Rover

Team Members:

James Joseph	Backend, Frontend, and Antipattern Developer
Samuel Lickteig	Backend and Database Developer
Alix Noble	Frontend Developer and Analytics Designer
Andrew Sand	Team Organization, CPR E 2880 Liaison
Owen Sauser	Code Composer Studio Implementation, CPR E 2880 Liaison

Team Website:
sdmay25-23.sd.ece.iastate.edu

Team Email:
sdmay25-23@iastate.edu

Revised:
4 May 2025

Executive Summary

Programming in C, especially in embedded contexts, can be a rather challenging language for students and novice programmers to learn, resulting in a higher barrier to entry and numerous struggles due to its intricate nature. For instance, compiler feedback can be cryptic, and deeply nested or confusing errors can arise that are formidable for beginners to debug.

The Code Critiquer System for the C Language and Embedded C is a web application and database that “critiques” submitted C source files for antipatterns or other poor programming practices. Additionally, it provides an architecture for instructors to manage their courses in the system to customize the experience and receive feedback or analytics regarding student usage of the tool. As a continuation of team sdmay24-34's work, the project is targeting usage in the CPR E 2880 (Embedded System I: Introduction) course at Iowa State University, where students will be able to upload their code for critiquing, and teaching assistants and instructors will be able to create assignments, select which antipatterns to test for, manage student user accounts, and view usage analytics. The project's overarching goal in this scope is to improve student learning while reducing the workload on instructors and teaching assistants, resulting in a significantly enhanced course experience.

The design must be easy to use by both students and staff, and the generated feedback must be returned in a timely manner and understandable to beginner programmers. Additionally, the project must be well documented and modular to support further programming languages and Iowa State courses in the future. The design's frontend web application is what users will interact with, allowing them to upload files to check, view the generated feedback, and more. It is written in HTML and CSS, with Bootstrap assisting with the appearance, and routing is handled by the Flask framework. Python code also serves dynamic data using templates. As for the project's backend, a database maintained using MySQL stores all of the system's information. Finally, the server sandboxes each critiquer run using Docker, ensuring a clean and consistent execution environment.

As this project is a continuation from team sdmay24-34's, most of the core frontend and backend design is already implemented, but it required refinement and various improvements before it is ready to be used in CPR E 2880. The team has made great strides in bringing it closer to a test run during a full Iowa State semester, getting the project to a minimum viable product state, but there is still plenty more that can be improved or implemented in the future.

Learning Summary

Development Standards & Practices Used

- Software Practices
    - Version Control - Project uses GitLab to organize workflow and ensure all developers can contribute simultaneously on different features
    - Continuous Integration and Continuous Deployment - Ensuring that the published build is up to date and deployed to a remote server
    - Testing - Running tests to ensure the system is stable and secure
- ABET Criteria
    - Applying principles of engineering, mathematics, and science
    - Identifying, formulating, and solving complex engineering problems
    - Communicate effectively with a wide range of audiences
- IEEE and ISO Engineering Standards (See section 2.2 for more details)

Summary of Requirements

- The critiquer must be easy and quick to use with a clean user interface
- The critiquer must catch antipatterns in C programs and in embedded contexts
- Generated feedback must be useful, concise, and aid the user in learning
- Usable in different contexts by students, teaching assistants, and CPR E 2880 instructors at Iowa State University (Principle of Least Privilege)
- Usage statistics must be aggregated and presented in ways useful to instructors for improving the CPR E 2880 course

Applicable Courses from Iowa State University Curriculum

- COM S 1850: Introduction to the C programming language
- COM S 3090: Team project management and web application development
- COM S 3170: Understanding how software is tested
- COM S 3270: Deepening knowledge of C
- COM S 3630: Understanding and managing database systems
- CPR E 2880: Embedded Systems and the primary concern of this project

New Skills/Knowledge acquired that was not taught in courses

- Abstract Syntax Trees
- Flask
- Regular Expressions
- XPath

Table of Contents

List of Figures and Definitions

Definitions:

Antipattern: A distinguishable error, bad practice, or flaw in written code. For example, an antipattern can manifest as a syntax error, stylistic inconsistency, inefficient structure, or compiler error.

Code Composer Studio (CCS): The software used in Iowa State's CPR E 2880 course developed by Texas Instruments to create and build programs for integrated circuits. It is used in this project as an optional part of the critiquing pipeline.

CyBot: The specialty built vacuum-like robots used in Iowa State's CPR E 2880 class. Assignments and labs involve writing code that will be executed on one.

Regular Expression: A series of characters used to identify and select patterns in a plaintext file or input. It is used in this project to search for specific patterns in code to perform static analysis.

TIVA TM4C123GH6PM Board: The microcontroller developed by Texas Instruments used in the CyBots. Students write code for these boards in CPR E 2880.

XPath: An expression language primarily used to explore files of XML format. It is used in this project to traverse and search the abstract syntax tree representations of code to perform static analysis.

List of Figures:

List of Tables:

1.0 Introduction

1.1. Problem Statement

The C programming language can be a rather tricky subject for students, beginner programmers, and even experts to comprehend and learn successfully. Compared to other programming languages, C can often be considered to have none of the "training wheels" or safeguards common with the others. However, the language is still widely used in several fields, such as operating or embedded systems, due to its efficiency and more nuanced control over computer hardware. Therefore, C is still an essential topic for most computer and software engineers. The overarching goal of this project is to reduce the barrier to entry for the language in an academic context tailored explicitly for the CPR E 2880 course at Iowa State University by improving upon the work done by team sdmay24-34. The created software system will allow students to upload their code to be automatically analyzed and have constructive, easily understood feedback generated. Furthermore, instructors and teaching assistants will be able to view and collect analytics about what students are submitting to the tool, allowing for better identification of problematic topics or poor practices. This software solution will also aid in reducing the workload on instructors and teaching assistants in addition to the response times to answer student programming questions.

1.2. Intended Users

Since the Code Critiquer System for the C Language and Embedded C project is targeting usage for Iowa State's CPR E 2880 course, three primary user groups have been identified:

User Group #1: Students

<u>Description:</u> CPR E Students are usually rather busy with their other classes and work, so they may not be available at the same times as the CPR E 2880 teaching assistants or professors. The students typically have some knowledge of the C programming language but need more in-depth or nuanced comprehension. They can frequently run into errors that they need help understanding.

<u>Needs Statement:</u> CPR E students need a way to get fast and reliable feedback on their work when teaching assistants or professors are unavailable.

<u>Benefits:</u> This product would allow the students to get feedback anytime and anywhere. It would describe to them the issues they are encountering and how they may fix or understand them. The feedback provided by the software solution will allow students to learn what mistakes they are making, why they are considered bad practice, and how to correct them.

User Group #2: Teaching Assistants
Description: Teaching assistants typically have a busy schedule, especially when working in large, lab-based courses such as CPR E 2880. They usually have the information a student may need to progress successfully but lack the ability to get to and help all students promptly.
Needs Statement: CPR E 2880 teaching assistants need a way for students to troubleshoot problems without their direct help.
Benefits: This product would give teaching assistants a lighter workload and allow them to focus more on their classes, work, or research. Furthermore, this software solution will significantly reduce the need for teaching assistants to answer trivial or recurring questions, allowing them to focus on giving students more complex or conceptual assistance.

User Group #3: Instructors
Description: Instructors design and create assignments and field questions from students regarding these assignments. Though they have contact with students, it can be challenging to gauge how their students are doing in class due to a lack of one-on-one time with students and student feedback.
Needs Statement: Instructors need a way to compare many students' work against a common set of issues.
Benefits: This product would reduce the number of questions instructors get from students, freeing up more time, similar to the benefits teaching assistants will see. It would also allow them to see what issues students have most often, allowing them to tailor their curriculum to students' needs better.

2.0 Requirements, Constraints, And Standards

2.1 Requirements & Constraints

1. **Functional**
   - Provide automated feedback based on the antipatterns in the database
   - Compile and run C code
   - Allow professors to add and remove custom antipatterns to the database
   - Allow students to submit code files for critiquing
   - Add clarity to existing static C analysis tools
   - Allow instructors and teaching assistants to view analytics of student-analyzed code per assignment per section
   - Allow students to submit comments/questions regarding antipatterns found in uploaded code
2. **Resource**
   - Linux server for running the web application
     - 1 Core
     - 4GB Memory
     - 64GB Storage
     - Network interface card (NIC)
     - Public IP address with addressable port (Constraint)
     - Compatible with Python and program libraries (Constraint)
     - Compatible with libc (Constraint)
   - Git repository for project development (To manage code and run pipelines)
3. **Aesthetic**
   - Clean and clear to make it easy to understand and receive the information
     - No overlapping text
     - Scales to multiple resolutions appropriately (Targeting only desktop resolutions such as 1920x1080p, as mobile is not a prioritized platform)
     - Using a legible serif or sans-serif font that is at least 12pt in size
4. **UI/UX**
   - Takes less than 10 seconds to analyze and return feedback
   - Easy-to-click buttons
     - No stacking buttons on top of each other
     - Buttons give visual feedback when they are clicked
     - Can click anywhere on the button to register clicking it

5. **Maintainability**
   - The application must be well documented so it can be maintained and expanded upon after the team is done
   - Instructors should be able to add, edit, and remove assignments in the system to better fit the needs of CPR E 2880 for a particular semester
   - Teaching assistants should be able to edit assignments in the system given their account has received edit permissions from the instructor.
   - Database of antipatterns should be easily updatable, allowing the tool to be used for future applications

2.2 Engineering Standards

1. IEEE 1028-2008 - This standard relates to the project because it talks about reviewing code. Reviewing code and work from other teams is a significant part of this project, as the group needs to build off of what both the prior senior design team and the Michigan Tech University.
2. IEEE 2675-2021 - This standard covers the concepts of reliably, securing, and safely building, packaging, and deploying applications in relation to DevOps. Since our project focuses on having both a frontend and backend, practicing efficient and safe DevOps will be critical to the group's success.
3. IEEE 1016-1998 - It covers the recommended practices for Software Design Descriptors, which are a medium used for conveying the structure of a software system. It will be important to be able to effectively and concisely communicate how the senior design project is structured to the clients and advisor.
4. ISO/IEC 9899:2018 - Explains how C code is made to compile and run. This will be a useful resource to compare code against. Most code that goes against these patterns will include antipatterns
5. ISO/IEC/IEEE 15288:2023 - This standard describes the terminology, concepts, and frameworks that deal with the lifecycle of a software system. It applies to bespoke and mass-produced systems, so it will be applicable to our project.
6. ISO/IEC TS 17961:2013 - The rules for secure coding in the C language are covered in this standard. Since our project heavily deals with identifying errors and antipatterns in C programs, this standard will be an excellent reference for a more memory-safe and cybersecurity standpoint.

3.0 Project Plan

3.1 Project Management/Tracking Procedures

This project will use a hybrid management approach, drawing inspiration from both Agile and Waterfall. The overarching project is broken down into large Waterfall-like tasks where some must be completed sequentially, but not all. However, each of these tasks will be worked on in an Agile manner with "sprints" taking place each week, consisting of a team meeting, advisor meeting, and planning goals to be completed for the next week. In the spring 2025 semester of Iowa State University's academic year, there will be "sprints" in the sense that functionality, feedback, and improvements will need to be worked on between each CPR E 2880 lab.

To aid in this development cycle, the team will use GitLab issues to track the progress of features and manage Git progress. With the consultation of the project's advisor and client, a system of priority management was devised, where each feature is categorized as part of the "Minimal Viable Product," "Minimum Marketable Product," or it is a "Stretch Goal." These priorities are reflected as labels on GitLab issues, allowing for members to better select high priority features or improvements to work on. For communication between team members, the project advisor, and members from the prior senior design group, the team will continue to use Discord, as it allows for quick, efficient bursts of communication that do not rely on availability. A shared Google Drive is also used to allow members to collaborate simultaneously on assignments and documentation, and it houses the meeting minutes in case a member is absent from a meeting or critical discussion for any reason.

3.2 Task Decomposition

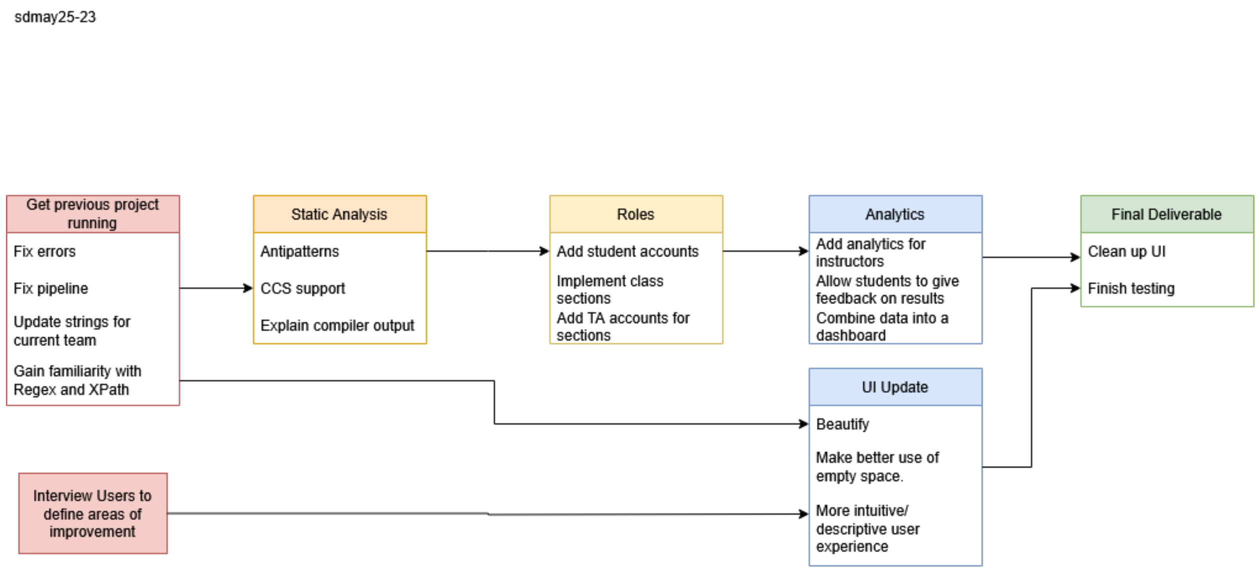

sdmay25-23
Get previous project running
Fix errors
Fix pipeline
Update strings for current team
Gain familiarity with Regex and XPath
Static Analysis
Antipatterns
CCS support
Explain compiler output
Roles
Add student accounts
Implement class sections
Add TA accounts for sections
Analytics
Add analytics for instructors
Allow students to give feedback on results
Combine data into a dashboard
Final Deliverable
Clean up UI
Finish testing
UI Update
Beautify
Make better use of empty space.
More intuitive/ descriptive user experience
Interview Users to define areas of improvement

Fig. 1 Task Decomposition Chart

1. Get the previous project running
   a. Fix errors
   b. Fix pipeline
   c. Update strings for current team
   d. Gain familiarity with Regex/XPath
2. Static analysis
   a. Antipatterns
   b. CCS support
   c. Explain compiler output
3. Roles
   a. Add student accounts
   b. Implement class sections
   c. Add TA accounts for sections
4. Finish prototype
   a. Analytics
      i. Add analytics for instructors
      ii. Allow students to give feedback on results
      iii. Combine data into a dashboard
   b. UI update
      i. Beautify

ii. Make better use of space

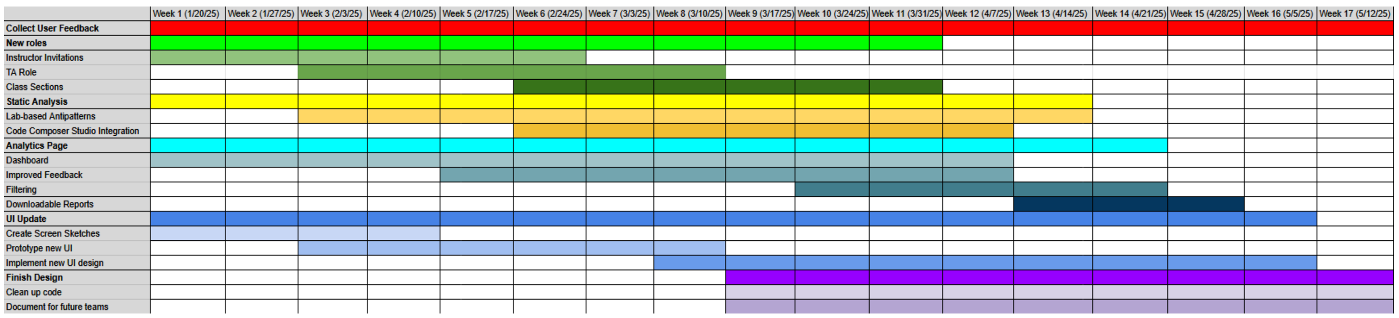
iii. More intuitive/descriptive user experience

5. Final deliverable
    a. Clean up UI
    b. Finish testing

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

1. The following functions of the previous project will work with zero critical errors (Errors that prevent the program from continuing)
    a. Create instructor account
    b. Login as an instructor
    c. Create an assignment as an instructor
    d. Create antipatterns as an instructor
    e. Upload unit tests as an instructor
    f. Upload project as a student
    g. Identify antipatterns in uploaded code based on antipatterns for the assignment
    h. Identify runtime and compile time errors based on tests uploaded for the assignment
2. Code will be compiled by Code Composer Studio and output parsed
3. Create at least one antipattern relevant to each CPR E 2880 lab
4. Increase user satisfaction of UI based on survey of select CPR E 2880 students
    a. Have a user and professor satisfaction of over 80%
    b. Determinant by polling of students and instructors

3.4 Project Timeline/Schedule



	Week 1 (1/20/25)	Week 2 (1/27/25)	Week 3 (2/3/25)	Week 4 (2/10/25)	Week 5 (2/17/25)	Week 6 (2/24/25)	Week 7 (3/3/25)	Week 8 (3/10/25)	Week 9 (3/17/25)	Week 10 (3/24/25)	Week 11 (3/31/25)	Week 12 (4/7/25)	Week 13 (4/14/25)	Week 14 (4/21/25)	Week 15 (4/28/25)	Week 16 (5/5/25)	Week 17 (5/12/25)
Collect User Feedback	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
New roles	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩						
Instructor Invitations	🟩	🟩	🟩	🟩	🟩	🟩											
TA Role			🟩	🟩	🟩	🟩	🟩	🟩									
Class Sections						🟩	🟩	🟩	🟩	🟩	🟩						
Static Analysis	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨				
Lab-based Antipatterns			🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨				
Code Composer Studio Integration						🟧	🟧	🟧	🟧	🟧	🟧	🟧					
Analytics Page	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦			
Dashboard	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦					
Improved Feedback					🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦					
Filtering										🟦	🟦	🟦	🟦	🟦			
Downloadable Reports													🟦	🟦	🟦		
UI Update	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	
Create Screen Sketches	🟦	🟦	🟦	🟦													
Prototype new UI			🟦	🟦	🟦	🟦	🟦	🟦									
Implement new UI design								🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	
Finish Design									🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪
Clean up code									🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪
Document for future teams									🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪

Fig. 2 Gantt Chart of Project Prototype Timeline

3.5 Risks and Risk Management/Mitigation

- **Collect User Feedback**
    - Risk: Unable to contact and interview users - 0.1
    - Risk: User requirements gathering does not produce sufficient patterns to focus development - 0.3
- **New Roles**
    - Risk: Implementation of the new role hierarchy introduces new security faults - 0.7
        - Reasoning: Given that the original sdmay24-34 prototype only had one type of user and was never intended for a public test, there were lots of security questions left unanswered. The team needed to decide how best to implement more than just instructor users while keeping in mind best practices for security.
        - Mitigation Strategy: The team is employing the principle of least privilege when handling accounts, ensuring that they only interact with what is strictly necessary to each type. Additionally, constantly testing the system on the server will give the team a better idea of potential loopholes that might compromise account security.
- **Static Analysis**
    - Risk: Static analysis does not correctly identify errors - 0.4
    - Risk: Static analysis exhibits a degradation of accuracy and precision - 0.1
    - Risk: Code Composer Studio exhibits aberrant behaviors in command line form - 0.3
- **Analytics Page**
    - Risk: Implemented analytics do not satisfy instructor and teaching assistant needs - 0.4
- **UI Update**
    - Risk: UI Update worsens application appearance and usability - 0.2
    - Risk: Newly designed pages can not be integrated - 0.1
    - Risk: UI update decreases application accessibility - 0.1
- **Finish Design**
    - Risk: Prototype can not be deployed - 0.3
    - Risk: Prototype exhibits never-seen-before problems - 0.7

- Reasoning: This project is rather complicated with several interconnected pieces of functionality, so there is bound to be issues that the team can not feasibly account for.
- Mitigation Strategy: Prior to deploying a prototype, the team will need to thoroughly do testing and stress tests. Furthermore, user testing feedback will be extremely beneficial to identifying and correcting erroneous problems.

Through the project, the team encountered several risks that were navigated to reach the desired deadline. The first that was run into was the fact that the team could not get the previous team's code to function properly. Even though the repository was cloned successfully and all of the provided setup instructions were followed accurately, no team member could get the system to work properly. To remedy this, the team employed the identified mitigation strategy of contacting a member from sdmay24-34 while also attempting to track down the issue using the error messages that the system was giving. Through the combination of these two techniques, the team was eventually able to get the prior team's prototype operational, but a significant amount of time was lost due to this roadblock.

Additionally, another major risk that was identified at the start of the project came to fruition: the team could not get dynamic analysis operational. Between continued discussions with the project's advisor and client and research into the technologies needed to simulate or remotely execute the TIVA board used in CPR E 2880, it became clear that other functionality and fixes should take higher priority. While a decent form of dynamic analysis could have been implemented during the project's work time, the parties involved with this project agreed that that time was likely better spent getting the system as a whole significantly more user-friendly, fixing faults, and improving the static analysis side of things. So overall, the team's mitigation strategy was to deprioritize implementing dynamic analysis in favor of getting the system more operational for a prototype test run in CPR E 2880.

3.6 Personnel Effort Requirements

Task Name	**Estimated Person-Hours Required**
Get the previous project running	60 Hours
User requirements gathering	20 Hours
Dynamic analysis	80 Hours
Static analysis	120 Hours
UI update	60 Hours
Finish prototype	40 Hours
Get student/professor feedback	Unknown
Incorporate feedback	Unknown
Clean up deliverable	60 Hours
Total	**440 Hours + Unknown**

Table 1 Estimated Personnel Effort Requirements

Many of these estimates are rather rough. Knowing all the errors and problems still left in the previous program is impossible. On the static analysis side, many members are presently getting up to speed with regex and XPath. Based on what the previous team accomplished over the course of a year, it will be challenging to gauge how long it will take to get lab-specific antipatterns ready. As for the UI update, several team members have some experience developing a UI on this scale. This time estimate is based on a few code jams and projects. Finishing the prototype and cleaning up deliverables will focus on testing, and those times are based on experience with COM S 3090 and other development classes that the team members have taken. Additionally, the volume and complexity of implementing student and professor feedback are impossible to estimate precisely. It will ultimately rely on the experiences and amount of feedback given at that time.

Task Name	**Person-Hours Dedicated**
Get the previous project running	30 Hours
User requirements gathering	10 Hours
Implement user analytics	40 Hours
Static analysis improvements	60 Hours
UI update	15 Hours
Set up prototype server	20 Hours
Get professor feedback	3 Hours
Incorporate feedback	20 Hours
Clean up deliverable	10 Hours
Total	**208 Hours**

Table 2 Final Personnel Effort Requirements

There are many reasons why our final time estimate was less than the initial time estimate. The first reason is that the initial estimates were rather rough, intentionally giving a very generous estimate with the amount of time that would be needed. The second reason is that the team decided to cut out dynamic analysis from the scope of the project. This cut was to focus more on making a minimum viable product that could be used in CPR E 2880 labs, and the virtualized CyBot is just an extra feature that was not a major priority. And the last major difference was the UI update/cleaning up the deliverable. The team did not realize how much of the UI update could be generalized, which made it much easier to apply to each page, massively speeding up the process.

3.7 Other Resources Requirements

Technical Cost:

- A server from Iowa State's Electronics and Technology Group
- Energy Costs will be minimal
- Uses open-source software

Human Cost:

- Staff required to maintain system once completed
    - Either training teaching assistants or a specialist in the Iowa State Electronics and Technology Group
- Whoever enters the antipatterns needs to be knowledgeable of regular expressions

4.0 Design

4.1 Design Context

4.1.1 Broader Context

Area	**Description**	**Effects of C Code Critiquer**
Public health, safety, and welfare	How does your project affect the general well-being of various stakeholder groups? These groups may be direct users or may be indirectly affected (e.g., solution is implemented in their communities)	Students will be able to get feedback on their code at any time of day, instead of just when TA's and Instructors are available, reducing stress and uncertainty. TA's and Instructors are able to get more feedback on common student pitfalls.
Global, cultural, and social	How well does your project reflect the values, practices, and aims of the cultural groups it affects? Groups may include but are not limited to specific communities, nations, professions, workplaces, and ethnic cultures.	This product could be a double-edged sword when it comes to how it will affect the profession of software developers. Because on one hand it can make it easier for students to learn to code and identify mistakes. But on the other hand it could become a crutch to those students, making them reliant on it instead of thinking through a problem first.
Environmental	What environmental impact might your project have? This can include indirect effects such as deforestation or unsustainable practices related to materials manufacture or procurement.	The environmental impact should only be the energy cost of running the server. All the other costs are negligible because it uses hardware that Iowa State already has access to.
Economic	What economic impact might your project have? This can include the financial viability	There are no plans to monetize this product. The technical costs of running this after the initial setup is

	of your product within your team or company, cost to consumers, or broader economic effects on communities, markets, nations, and other groups.	just the energy costs, which will be covered through Iowa State University processes. The human cost is more prevalent. After the product is completed, it will need to be maintained, and it may need to be updated if CPR E 2880 changes enough. This would require training a teaching assistant, professor, or other employee to do so.

Table 3 Broader Context Table

4.1.2 Prior Work/Solutions

As the team is continuing the work of a previous senior design team, we were able to begin working through our design with a basic prototype at hand. The previous team was in frequent contact with Michigan Technology University, who had made several static code analyzers for other languages, which they used as an aid in their development. References to each prior work or solution can be found in section 8.2.

Previous Team's (sdmay24-34) C Code Critiquer [1]:

Advantages: Looking at a functional prototype has allowed the team to get a good understanding of how both regex and XPath are used in static analysis. It also provided a strong foundation to begin building new ideas and changes off of.

Shortcomings: The team was unfamiliar with Python, which the critiquer is coded in, at the start of the project. This shortcoming has hindered the understanding and learning of the code.

Michigan Technology University Critiquers [2]:

Advantages: Michigan Technology University has made multiple critiquers for several different programming languages, with functioning prototypes.

Shortcomings: As prototypes, they are not widely available for use. Also, the critiquers only use static analysis.

The team also spent time looking into code critiquers currently on the market, and found two to compare with.

PC-lint Plus [3]:

A commercial command-line tool that performs static analysis of C and C++, indicating issues or concerns in the code.

CodePal [4]:

A platform that provides several helpers and tools to assist developers. When looking into it, the main focus was on the code review portion of the platform.

Pros of Code Critiquer System for the C Language and Embedded C compared to others:

Designed to be used in CPR E 2880, this critiquer will be tailored to the course and embedded C programming as a whole. It will be reliable in finding the specific antipatterns that the professor is looking for. Furthermore, the explanations provided by the critiquer will be more beginner friendly.

Cons of Code Critiquer System for the C Language and Embedded C compared to others:

It requires more bespoke knowledge of its inner workings to maintain and a deeper understanding of regular expressions to add more antipatterns to the system.

4.1.3 Technical Complexity

The team's design expands upon the previous team's (sdmay24-34) code significantly. It is planned to add more antipatterns to the system based on what is being covered in the CPR E 2880 labs and the most common issues encountered in those assignments. This section of the project focuses substantially on leveraging XPath and regular expression technologies to represent and find antipatterns. Additionally on the static analysis side of things, Code Composer Studio (CCS), the software used to write code in CPR E 2880, is integrated into the system, allowing students to have their code compiled with it instead of GCC, resulting in more tailored feedback. CCS must be integrated in a seamless way that allows students to upload their code to the system’s GUI, have it be compiled by CCS, and then the system parses the compiler output of CCS, and then displays the curated results on the frontend back to the user.

The team is also adding a feedback page that the instructors and teaching assistants can use to see the most common antipatterns for each lab, among other statistics. This new functionality, along with the desired user interface overhaul, will require deep knowledge of frontend development, such as managing the router, writing accessible HTML and CSS, and hooking in third-party libraries for displaying charts.

Even further, the backend code of the system requires a lot of work from the previous sdmay24-34's progress given that their prototype only consisted of a single instructor type account. For this project, the team needs to expand on the number of account types to accommodate students and teaching assistants. Additionally, secure methods for handling credentials and logins must be implemented, ensuring a safe user experience. This implementation requires knowledge of cybersecurity measures and practices while adding them effectively to the system.

Overall, this project is rather technically complex, with frontend and backend development skills requirements, static code analysis, and connecting with external software and hardware. Each design section has unique challenges that must be overcome to ensure a successful outcome.

4.2 Design Exploration

4.2.1 Design Decisions

1. The first significant design decision the team had to make was if we wanted to reuse the previous team's code. Remaking it would allow us more flexibility and make it so we can develop the program with methods or languages that we are more familiar with. However, we decided against this, primarily because having a mostly working baseline allows us to spend more time on the product-adjacent tasks, such as integration into other systems or further functionality, without us having to spend time remaking the project's core. If we were to remake the project from scratch, that would consume most of the team's time, and it would be a significant risk on whether or not the final result would be improved over what the prior team accomplished, further pushing back the desired deadline for testing this system in a classroom environment.
2. Another decision we made was whether or not to include artificial intelligence in some form in this project. This could consist of having a large language model give answers about the 1,400-page long datasheet used in CPR E 2880 or

having it train on the course materials as a whole using something like CourseGPT(https://arxiv.org/pdf/2407.18310). Ultimately, it was decided not to involve artificial intelligence in this project's scope, as it could introduce unnecessary complications, false positives, and overall confusion given the technology's current maturity. Additionally, most team members have little to no experience working with such technologies, meaning that development would have been slow and inefficient. It should be noted, the client and advisor is open to considering the idea in future developments if it is determined artificial intelligence can be adequately implemented in a useful fashion.

3. Another design decision that was decided upon after much consideration is how we want to revise how the assignments are handled. The way sdmay24-34's prototype handles them is that the instructor will make an assignment, and then they will have to share the assignment code with their students. However, this approach makes it difficult to maintain the Principle of Least Privilege and track per-student analytics, both of which are integral requirements. To accommodate for this, we decided to have instructors import class sections and registered students, which they have authority over. This is an important decision because the highest priority behind the product working is having the system be something that the instructors would actively want to use in their sections, and if it is too inconvenient to set up the assignments, they will not want to use it.

Overall, these three key decisions were integral to the project's success. Each decision kept in mine the overarching goal of the project while keeping the scope within a reasonable and manageable size. When making the decisions, the team kept the project's requirements and constraints in mind so that none of them are violated, and the team's strengths can be tapped into, maximizing project success and completion likelihood.

4.2.2 Ideation

The decision to stick with the current code came with several potential options. Our team as a whole had little experience in python (which the previous team used) and jumping into a relatively complete project could be difficult.

Our first option was to stick with the previous team's work and spend some time getting to know the code. Even if we are unfamiliar with python, it shouldn't take long to gain a decent understanding of it.

Our second option was to build a new project from scratch using C, the language being critiqued by the system. This would theoretically make it easier to test the code being uploaded by students.

Our third option was to build a new project from scratch using Java, the language we were most familiar with. It wouldn't be as easy to critique C code, but the structure of the project would be more easy to build.

Our fourth option was to scrap the idea of a web application and spend our time making a plugin for Code Composer Studio, the IDE being used by the CPR E 2880 lab. This would mean that the students could directly critique the code without uploading it through a browser.

The final option we considered was making a plugin for Visual Studio Code, and having the CPR E 2880 lab switch to using it instead of Code Composer Studio. This is the most difficult option currently as it involves changing the lab structure.

4.2.3 Decision-Making and Trade-Off

We identified the pros and cons of the ideated approaches throughout a few meetings by first making a list of the options. Everyone brought forth their opinion, and we discussed which option would be easy to jump into, develop in a timely manner, and produce satisfactory results. With our client, Dr. Rover, wanting to see a prototype in the Iowa State Spring Semester, the option with the least risk came down to sticking with the current code. Starting fresh with a different language would be challenging to complete in time, considering we would both need to understand the project and create it all in one semester. While both the Code Composer Studio option and the Visual Studio Code option were appealing initially, they slowly lost favor after discussions continued with Dr. Rover and Dr. Jones (the other professor teaching CPR E 2880), and the project design continued being refined. Code Composer Studio is undergoing an overhaul in December 2024, so it is still being determined what that software will look like throughout development. Switching to Visual Studio Code would create some risk, as it has yet to be used in the CPR E 2880 labs. As such, we have decided to work on the current code and add to the previous team’s design, especially as it seems like the natural evolution of this project. However, we may work some aspects of the other options into the project in the future, depending on talks with our client.

4.3 Final Design

4.3.1 Overview

The system's design is a database with an accompanying website running on a central server. Users can interact directly with the system by logging into the website as a student, teaching assistant, or instructor, depending on their role in their class. As an overview, instructors create assignments that allow students to upload their code and have the system critique their work, providing helpful feedback for diagnosing programming problems or building better practices. Instructors can also manage multiple courses, sections, and users enrolled in those while having control over class assignment details and viewing analytics about the system's usage, such as how many students triggered a specific antipattern warning on a particular assignment. The backend database is used for storing system information and user data, while the frontend website is utilized by the users to access coding critiquing and stored information without directly accessing the database.

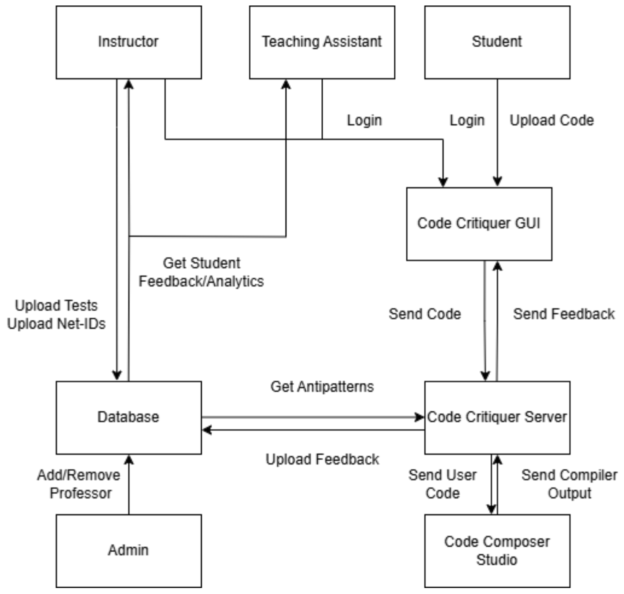
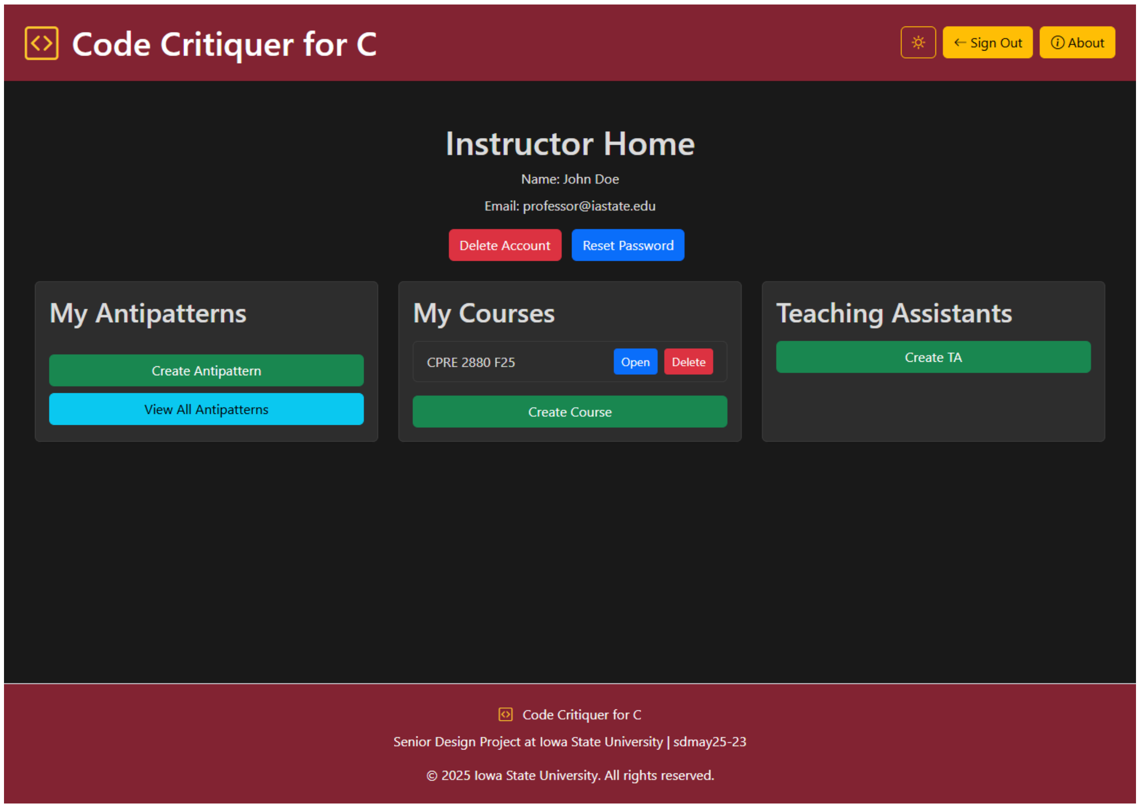


Fig. 3 System Overview Sketch

4.3.2 Detailed Design and Visuals

The project's design is a comprehensive web application with a frontend system of web pages that users will access to communicate with the backend database server that stores account, assignment, and analysis information. Uploaded student code is critiqued on the server using static analysis methods and, optionally, Code Composer Studio to generate and provide feedback to the user, which can then be used to help determine faults in the code or highlight good practices. Instructors and teaching assistants can view analytics and download reports about assignments, which pulls and aggregates data stored in the database from student usage.



Name: John Doe

Email: professor@iastate.edu

Fig. 4 The Instructor Home Page

<u>Frontend Graphical User Interface:</u>

The frontend web application's graphical user interface is built using HTML, CSS, JavaScript, Jinja, Bootstrap, Popper, and Flask so that multiple users can access it simultaneously and navigate pages with the correct information as expected. Users use the frontend web application to interact with the backend database, meaning that users should not be directly accessing the database under normal usage. For instance, the frontend GUI receives and displays the feedback from the critiquer server after a student submits code to be analyzed. As another example, Fig. 4 shows the Instructor Home Page, the first page displayed after an instructor logs into the website. From this screen, an instructor can access all of the necessary functionality that they need.

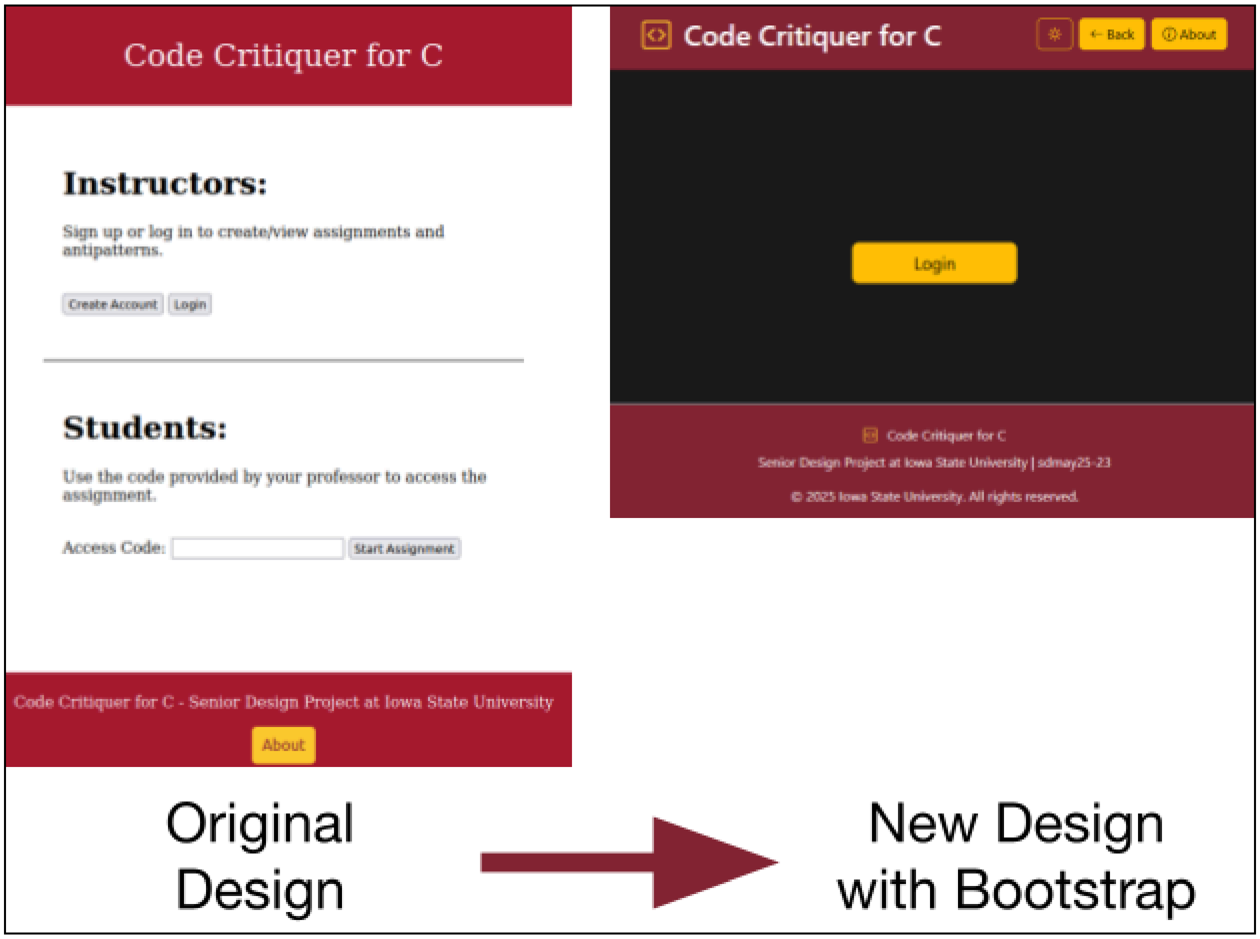

Code Critiquer for C
Instructors:
Sign up or log in to create/view assignments and antipatterns.
Create Account
Login
Students:
Use the code provided by your professor to access the assignment.
Access Code:
Start Assignment
Code Critiquer for C - Senior Design Project at Iowa State University
About
Code Critiquer for C
Back
About
Login
Code Critiquer for C
Senior Design Project at Iowa State University | sdmay25-23
© 2025 Iowa State University. All rights reserved.
Original Design
New Design with Bootstrap

Fig. 5 Comparison of the Landing Page (Pre vs. Post Bootstrap)

The team decided to use Bootstrap to ensure that the frontend looks presentable and visually appealing, in contrast to the rudimentary raw CSS that the system originally used. For instance, Fig. 5 compares the original look and the new design of the frontend. Bootstrap allowed the team to rapidly implement a significantly better-looking user interface while leveraging some quality-of-life features that users expect from a modern web application, such as toggling between light and dark modes.

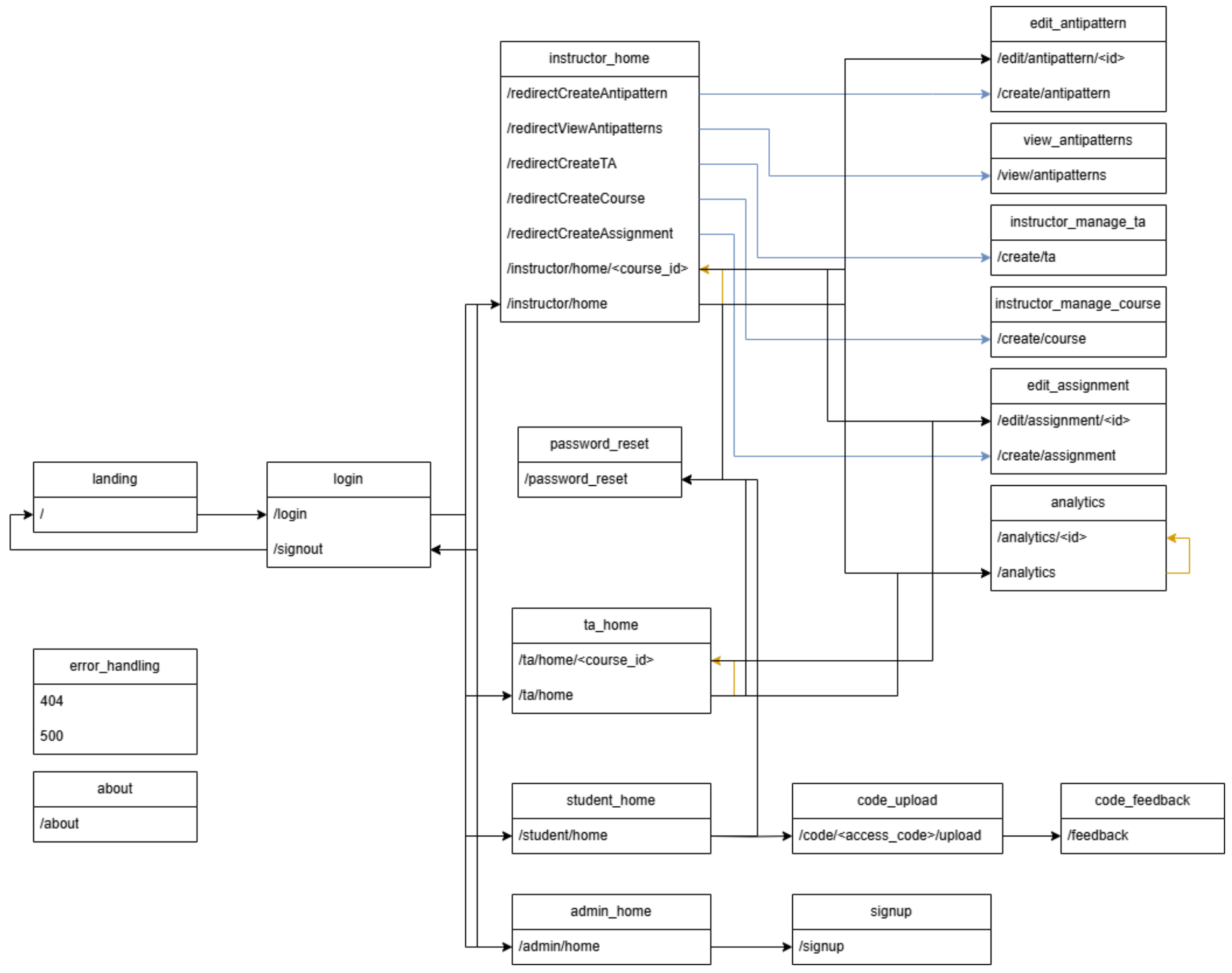


Fig. 6 Page Flow Diagram

Fig. 6 is a diagram of the page flow through the GUI, as experienced by a user. It depicts how each page in the system is connected and how it can be accessed. The routing of pages in the web application is handled through Flask. Ultimately, a frontend framework was decided against since the system's response time is a critical requirement, and there are not many elements in the project that would benefit from being dynamic. Instead, Jinja is used to template the pages, allowing for the appropriate information to be loaded when a user accesses the system.

Backend Database:

The backend server stores all of the information regarding the operations of the system and it will be hosted in a single location, serving each of the users connecting using the frontend web application. The database facilitates the logging in and registering of users while also storing necessary data for operation, such as antipatterns and analysis feedback. Students will be able to submit their code using the frontend GUI to the database, which will then be sent to the critiquer for running. The database will then receive feedback information from the critiquer once the tests have finished running. It will be maintained using MySQL with the following tables to store the information.

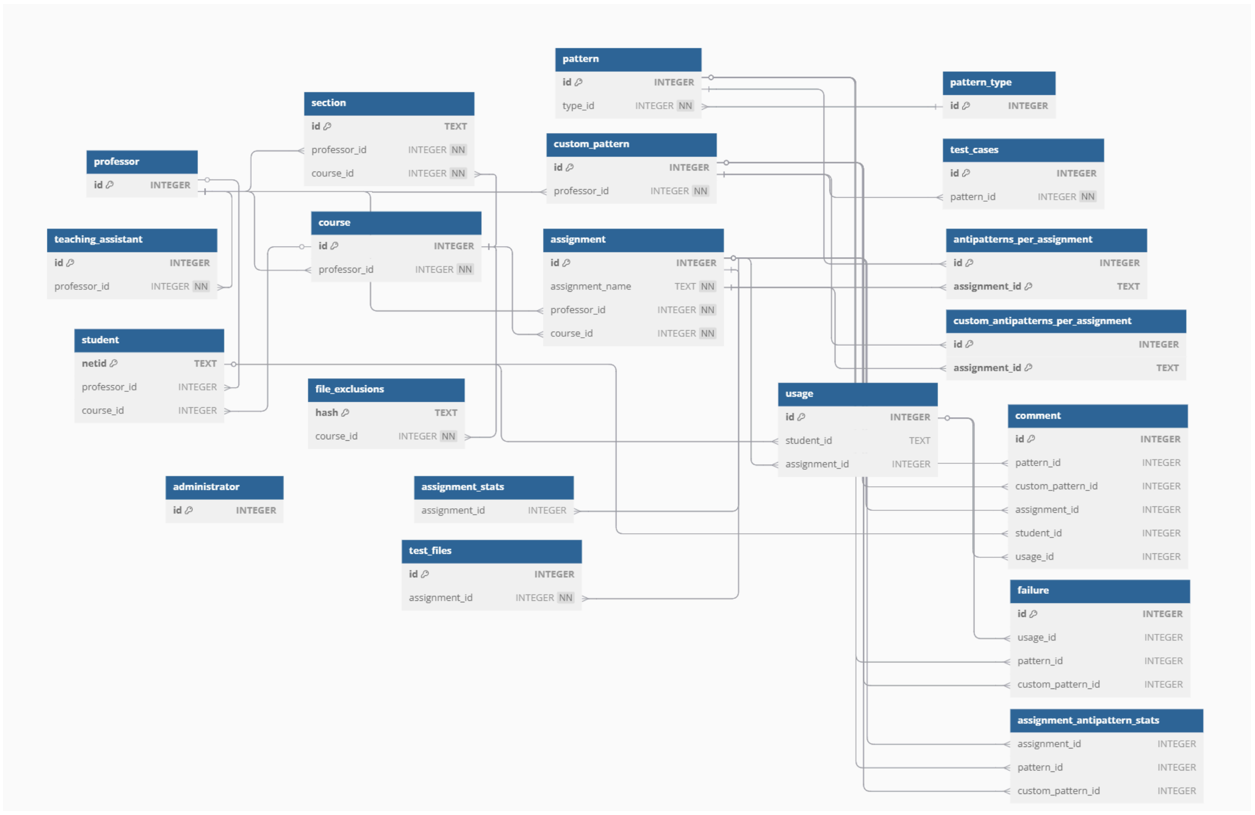
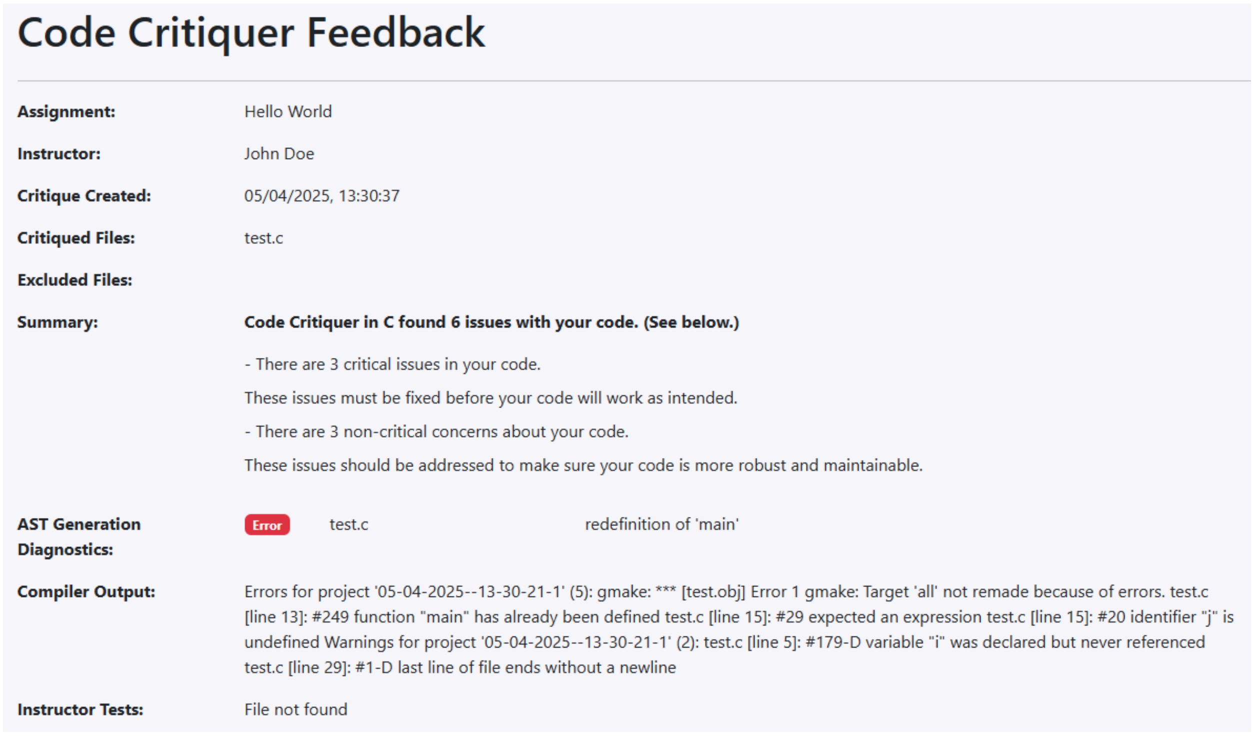


Fig. 7 Database Table Diagram

<u>C Critiquer:</u>

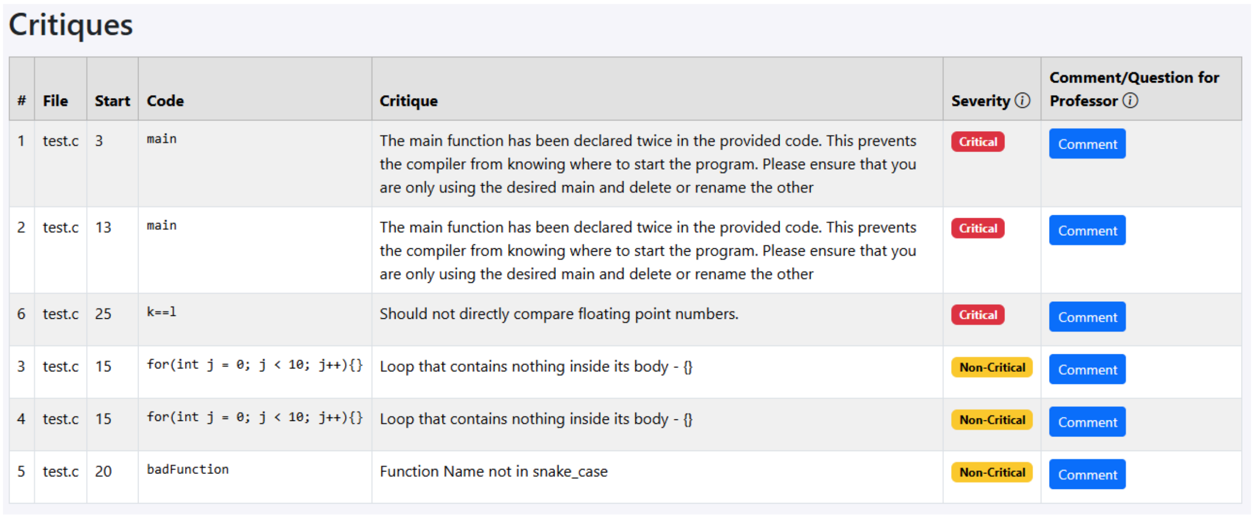
Additionally, there is a critiquer in the project that houses the sandboxing and C critiquer written in Python that is used for testing uploaded student code, generating feedback based on analysis, and getting antipatterns for testing from the database. The critiquer uses GCC to partially compile the submitted code, which allows an abstract syntax tree to be derived. From this point, regular expressions are leveraged to search for antipatterns in the AST, such as having two main functions defined. Regular expressions are also used to detect some antipatterns in the uncompiled raw code, most of which are targeted at syntax problems. Optionally, the user can also choose to have their code compiled using Code Composer Studio, the development software used in CPR E 2880. When a student decides to compile using CCS, their submitted code is compiled using the same libraries used in class. This behavior allows for accurate compiler error or warning messages, which are parsed by the system, making them easier for students to understand. After analysis, the virtualized system generates feedback and displays it to the user through the frontend interface. Fig. 8 and Fig. 9 show an example set of critiques.



Code Critiquer Feedback

Assignment:	Hello World
Instructor:	John Doe
Critique Created:	05/04/2025, 13:30:37
Critiqued Files:	test.c
Excluded Files:	
Summary:	**Code Critiquer in C found 6 issues with your code. (See below.)** - There are 3 critical issues in your code. These issues must be fixed before your code will work as intended. - There are 3 non-critical concerns about your code. These issues should be addressed to make sure your code is more robust and maintainable.
AST Generation Diagnostics:	Error test.c redefinition of 'main'
Compiler Output:	Errors for project '05-04-2025--13-30-21-1' (5): gmake: *** [test.obj] Error 1 gmake: Target 'all' not remade because of errors. test.c [line 13]: #249 function "main" has already been defined test.c [line 15]: #29 expected an expression test.c [line 15]: #20 identifier "j" is undefined Warnings for project '05-04-2025--13-30-21-1' (2): test.c [line 5]: #179-D variable "i" was declared but never referenced test.c [line 29]: #1-D last line of file ends without a newline
Instructor Tests:	File not found

Fig. 8 Example Critiquer Feedback Overview



Critiques

#	File	Start	Code	Critique	Severity	Comment/Question for Professor
1	test.c	3	`main`	The main function has been declared twice in the provided code. This prevents the compiler from knowing where to start the program. Please ensure that you are only using the desired main and delete or rename the other	Critical	Comment
2	test.c	13	`main`	The main function has been declared twice in the provided code. This prevents the compiler from knowing where to start the program. Please ensure that you are only using the desired main and delete or rename the other	Critical	Comment
6	test.c	25	`k==1`	Should not directly compare floating point numbers.	Critical	Comment
3	test.c	15	`for(int j = 0; j < 10; j++){}`	Loop that contains nothing inside its body - {}	Non-Critical	Comment
4	test.c	15	`for(int j = 0; j < 10; j++){}`	Loop that contains nothing inside its body - {}	Non-Critical	Comment
5	test.c	20	`badFunction`	Function Name not in snake_case	Non-Critical	Comment

Fig. 9 Example Breakdown of Critiques

4.3.3 Functionality

The project's design has three user groups in mind, each of which will interact with the system in slightly different ways.

1) CPR E 2880 Students

In the system, the CPR E 2880 student user group can log into the web application, reset their password, select a specific assignment, and submit code to be analyzed. Once submitted to the system by a student user, their code will be analyzed and feedback generated, which the student can then use to learn and improve their programming skills or build better practices.

2) CPR E 2880 Professors

The CPR E 2880 professor user group will be able to log into an account on the system's web application, and from there, they will be able to set up user accounts for teaching assistants or students, create, edit, or delete assignments or courses, assign students and teaching assistants to various sections, edit or add antipatterns to test for, view analytics from student submissions, and download usage reports. Pages to accomplish these functions will be hosted through the web application, but all of the user, assignment, and class information will be hosted on the backend server. Instructors can view statistics and analytics about students' critiquer submissions for a particular assignment. This data will aid instructors in determining problem areas for students, allowing them to focus on concepts prone to causing struggles, improving the CPR E 2880 class overall.

3) CPR E 2880 Teaching Assistants

The teaching assistants interacting with the system will utilize it similarly to the professors user group. The teaching assistants will be able to log into the web application using an account provided to them by a professor, and they will also be able to view student analytics, edit assignments (provided they have been given editing permissions by the professor), and download student reports. However, they can not add or remove assignments, edit student accounts, or edit courses. Additionally, teaching assistants can not edit antipatterns, as that is a functionality restricted to professors, even though they can change which antipatterns are checked for in an assignment.

4.3.4 Areas of Challenge

Throughout the course of the work period, this project faced several challenges over most aspects. Firstly, a significant chunk of the challenge areas resulted from variability in the project's requirements and user or client needs. When originally planning and setting the scope of the code critiquer, it was decided that dynamic analysis using either a simulated or remotely accessed TIVA board would be implemented, allowing students to upload their code to actually be run. However, as the project continued, it was eventually reevaluated that other functionality or fixes should be prioritized instead given the complexity and logistics of dynamic analysis. The client and advisor preferred to have more functionality and a smoother user experience at the expense of the dynamic analysis implementation, so the team faced a challenge of reprioritizing work items and time allocation. Consistent communication and an unwavering focus on keeping user needs and use cases at the forefront of the design ensured that the team was able to overcome this.

Additionally, there were several challenges faced from the technical aspects of the project. The code critiquer is a rather interconnected and complex system, so it took time for most team members to become acquainted with all of the technologies and concepts involved, especially since the prototype provided by the prior team had difficulties running. These challenges were mitigated or overcome through communicating findings or knowledge within the team, reaching out to members from the prior team, and meeting weekly with the project's advisor and client. Several of the technical decisions that came with reusing the last team's prototype also raised challenges, such as issues retrieving symbols in the static analysis, which was solved by tracking down the root of the problem and using a different version of the abstract syntax tree library, or a lack of reactivity or visual appeal to the frontend since it was written in plain HTML and CSS, which was overcome by implementing Bootstrap and using JavaScript combined with a charting library to get more reactive elements.

4.4 Technology Considerations

For the most part, the team continues to build off the designs and technological considerations of the prior senior design group, sdmay23-34. This stance means that in the current design stage, the frontend web application is made using HTML, CSS, JavaScript, and Bootstrap with Flask as a routing solution, and the backend server is maintained using MySQL. Additionally, the critiquer system itself is built using the ones designed at Michigan Tech University as a starting point, but it is using Python instead of the language it is analyzing, which is C in this case.

While the team has discussed moving toward more sophisticated solutions, such as a frontend framework, the primary goal of the design currently is to build off the prior work and create a working prototype. Additionally, there were concerns about working on the Python critiquer since many of the team members had little to no experience with the language. However, translating it to a different technology was ultimately decided against since it would require rewriting the system while using precious time. Additionally, since Python is one of the easier to learn languages for programmers, the team members can pick it up quickly while also learning a sought after skill.

5.0 Testing

5.1 Unit Testing

For automated unit testing, two main components need to be checked: regex/XPath antipatterns and unit tests for dynamic testing. For the regex/XPath antipatterns, multiple files will be uploaded, and the resulting found antipatterns will be checked against the expected list. Dynamic testing will use previous students lab solutions, after professor review, to check that all tests pass correctly. After that, modifications will be made to the lab to check that each test fails individually, and a random combination of tests will be checked to assure that they fail in the correct order. Of note, an error such as two mains will cause the program not to compile, so runtime errors will not be checked.

5.2 Interface Testing

As mentioned above, site flow will be manually tested to ensure correctness. Furthermore, the user interface will also be compared against common internet standards for accessibility and usability, ensuring that our project can be used by a wide audience conveniently.

5.3 Integration Testing

Example files will be sent through the API on the website to make sure that both the static and dynamic analysis portions are functioning properly. For static analysis, it will check that it only gives results for enabled and triggered antipatterns. Tests regarding dynamic analysis will make sure that the docker container is run properly, the makefile compiles properly, and results get parsed and passed back to the website.

5.4 System Testing

For the system test, the following steps will be taken:

1. Create a new professor
2. Login as the professor
    a. Incorrect password fails
    b. Incorrect email fails
    c. Correct password succeeds

3. Create a new antipattern
4. Create a new assignment
5. Select specific antipatterns for the new assignment
6. Upload a unit test for the new assignment
7. Upload example code to the new assignment
    a. Bad file format fails gracefully
    b. Correct antipatterns are checked (including the new one)
    c. Correct tests pass and fail

5.5 Regression Testing

To ensure that features don't regress, a pipeline will be run whenever code is merged into the main branch. This pipeline will fail if any of the aforementioned tests fail. This process happens in the project’s GitLab repository, ensuring that any code a team member pushes is tested this way.

5.6 Acceptance Testing

For acceptance testing, the team is in close and frequent communication with the project advisor/client and other faculty or teaching assistants closely related to the CPR E 2880 course. The team meets weekly with Dr. Diane Rover, the project advisor and one of the instructors for CPR E 2880, to discuss the design and its effectiveness, allowing for continuous evaluation, testing, and feasibility assessment. Furthermore, in the coming Iowa State Spring Semester, a prototype of the critiquer system will be used as an optional resource, allowing students to test the design first-hand, provide feedback, and generate usage telemetry, which will be invaluable for reflecting on and refining the design. Continuous communication with CPR E 2880 teaching assistants, instructors, and students while also supplying a prototype for feedback will allow the team to test the design's acceptance rigorously, ensuring that the functional and non-functional requirements are being fulfilled satisfactorily.

5.7 Security Testing

Since user data is being stored for this project, security is an utmost concern. First, SQL queries are being sanitised to ensure that injection can not be possible. Pip will be used to scan for out-of-date and vulnerable libraries. Nmap and burpsuite will scan for simple web exploits. Aside from that baseline testing, cybersecurity students and professionals will have a chance to review and analyse the project for vulnerabilities. This process will happen after every major version update.

5.8 Server Testing

To better test the code critiquer system in an environment closer to a production state, the team utilized a remote server where the code was continuously pushed and deployed. The team used a server the Iowa State Electronics and Technology Group provided, which needed to be requested early in the work process. Anytime code was merged into the master branch of the project's GitLab repository, it would then be deployed to the server through the CI/CD pipeline. By leveraging this testing server, the team could better test and see the system running in a close-to-production environment. It also provided a good visual for the client and advisor to view and experiment with, allowing for more specific and fast feedback.

5.9 Results

Overall, testing has ensured the system operates smoothly and as intended throughout the project's development. The bulk of the testing during development has been carried out through the GitLab CI/CD or on the ETG-provided server, both of which have been immensely helpful. While not perfect, the CI/CD pipeline has helped catch faults before the code is merged into the main branch, removing the possibility for those issues to root deeper into the project and become larger issues later. Given that this project will need to be used by multiple users simultaneously with minimal performance impact, as stated in the requirements, testing on a remote server with a production environment has been incredibly useful for testing those aspects while getting real-time results.

6.0 Implementation

6.1 Design Analysis

At the end of the project, almost everything as described in the final design has been implemented as intended, with some exceptions noted later. Given that the project is a continuation of sdmay24-34's work, the team required time to become familiarized with their prior accomplishments and the system left behind. Even so, the team managed to implement Bootstrap to overhaul the user interface (An example of which can be seen in Fig. 10), fix bugs from the previous prototype, integrate optional Code Composer Studio compilation support, improve the static analysis pipeline, add more antipatterns to the system, implement a user account hierarchy, add basic analytics and downloadable reports, add courses instead of just assignments, added the ability to exclude instructor-provided files from analysis, and created account registration. With weekly meetings and constant communication with the client and advisor, they have expressed approval of the accomplished work and a desire to begin using it for a trial run during the next semester of CPR E 2880.

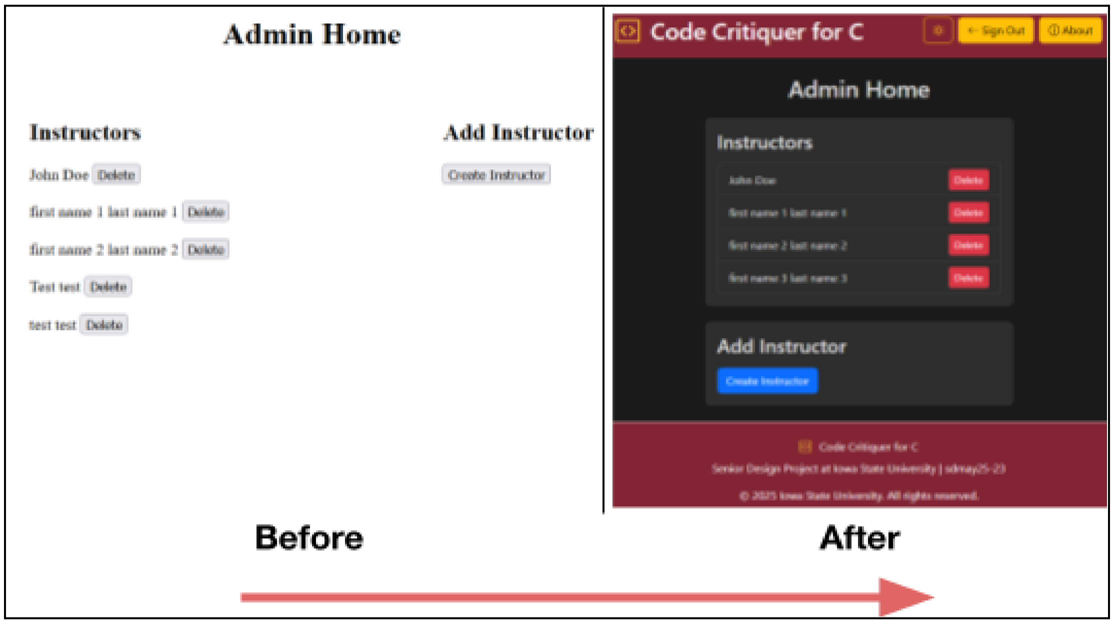

Admin Home
Instructors
Add Instructor
John Doe Delete
first name 1 last name 1 Delete
first name 2 last name 2 Delete
Test test Delete
test test Delete
Create Instructor
Code Critiquer for C
Sign Out
About
Admin Home
Instructors
John Doe Delete
first name 1 last name 1 Delete
first name 2 last name 2 Delete
first name 3 last name 3 Delete
Add Instructor
Create Instructor
Code Critiquer for C
Senior Design Project at Iowa State University | sdmay25-23
© 2025 Iowa State University. All rights reserved.
Before
After

Fig. 10 A Before and After Comparison of the About page

The original design conceived at the end of the first semester of working on the project included a few features that did not make it into the final design or were shelved for various reasons. The primary and arguably largest of these features is dynamic analysis, which would have complemented the static analysis portion of the critiquer, allowing students to get feedback from running their code on a virtualized CyBot. As the project progressed, it was decided, with coordination with the advisor and client, that the scope should be refined to focus on providing a more robust system. Dynamic analysis is a significantly more complex problem to approach, and all parties involved agreed it would be best to prioritize getting more functionality of the system implemented before worrying about dynamic analysis.

Given that the concrete requirements for the project were a moving target, there were a couple of aspects that the team feels could have been implemented better given more time. For instance, more testing and refinement of the web interface would be beneficial, given that even though it is a minor issue, things like the back button sometimes logging the user out can cause user frustration. Additionally, the analytics side ran into more challenges than anticipated, especially with providing charts or filters that instructors can use. These problems could have been better avoided if a frontend framework had been implemented, but the team opted to stick without one for performance and complexity concerns. While this approach worked well for this design, it was not without drawbacks as well, and future iterations might consider implementing a framework if dynamic elements like the analytics pages are to be expanded. However, the team is still proud of the work accomplished, given that a lot was still done in the short period of time to work on the project.

7.0 Ethics and Professional Responsibility

7.1 Areas of Professional Responsibility/Codes of Ethics

Area of Responsibility	**Definition**	**Related NSPE Code of Ethics**	**How Our Team Interacts with this Area of Responsibility**
Work Competence	Completing your work with a professional level of competency and timeliness.	II.2.a: Engineers shall undertake assignments only when qualified by education or experience in the specific technical fields involved.	When we make a feature we try to make sure it is thoroughly tested before pushing it to main.
Financial Responsibility	Being conscientious about the costs required to not just launch but maintain the project.	II.4.c: Engineers shall not solicit or accept financial or other valuable consideration, directly or indirectly, from outside agents in connection with the work for which they are responsible.	We are keeping the costs in mind while developing. While the upfront costs might be low, the cost to maintain and expand upon the project in the future might add up.
Communication Honesty	Being realistic about the capabilities of the team and the product to the stakeholders/users.	III.3: Engineers shall avoid all conduct or practice that deceives the public.	We are staying honest with our client about our progress and regularly sending them progress reports.
Health, Safety, Well-Being	To put the wellbeing of mankind above every other aspect of the product or project.	III.2.a: Engineers are encouraged to participate in civic affairs; career guidance for youths; and work for the advancement of the safety, health, and	We plan to have the project to make the website compatible with text readers for the visually impaired.

		well-being of their community.	
Property Ownership	Handling user-provided materials and information with due care and respect.	I.4: Act for each employer or client as faithful agents or trustees.	We are handling user data with care. Making sure that the TA and instructor accounts are secure to make sure students can not cheat.
Sustainability	Being conscientious of the impact the project can have on the environment, both directly and indirectly.	III.2.d: Engineers are encouraged to adhere to the principles of sustainable development in order to protect the environment for future generations.	The only thing for sustainability we have to account for is the energy cost of the server because all the hardware we use is stuff that Iowa State already has.
Social Responsibility	Making sure that the product is a net good for society without negatively impacting some.	II.1.e: Engineers shall not aid or abet the unlawful practice of engineering by a person or firm.	We want to make sure that students are using the product as a tool to supplement learning instead of something to replace their learning.

Table 4 Area of Professional Responsibility/Codes of Ethics

Overall, the team is performing well in the property ownership category, especially given the number of parties involved with the project. Since the critiquer will need access to CPR E 2880 lab answers, the team is ensuring that only the appropriate people have access to those materials, as it could be damaging if they were publicly released. Furthermore, CyBot-specific information is also carefully handled by keeping it contained within the scope of the critiquer and not giving out access to code repositories that the team is not in ownership of. The team communicates frequently with and receives feedback from the project client, ensuring that it is known which information is considered sensitive. The principle of least privilege and access is heavily utilized to ensure that only the correct users have access to only the subset of information that they need.

The area of professional responsibility that our team could improve on is work competence. Due to some unforeseen issues with the previous team's code we started to fall behind on our deadlines. And once we fell behind the deadlines we never adjusted them to account for this, so we fell even further behind. In the future we will be talking about our progress and how we are keeping up with the deadlines and adjusting them accordingly.

7.2 Four Principles

	Beneficence	Nonmaleficence	Respect for Autonomy	Justice
Public health, safety, and welfare	Helps improve stress levels of students and TAs	Aim to reduce student reliance on critiquer	Allow students to use as much as needed	Provides a resource for students to assist with learning
Global, cultural, and social	Addresses the needs of instructors and TAs	Available for all students in CPR E 2880	Respects cultural practices	All user types benefit from implementation
Environmental	Design uses existing hardware and will receive energy from Iowa State	Design does not require more manufacturing	Will provide user choice between just static analysis compiling with CCS as well	Allow students off campus to access hardware without producing more
Economic	Design is free for students	Design would not be disruptive	Will allow users to use only static analysis with GCC, which takes	Free resource for people who can not attend office hours

			less time and energy to compile and critique	

Table 5 Four Principles Table

One broader context-principle pair that is important to our project is public health, safety, and welfare paired with beneficence. A main goal of the critiquer is to provide timely feedback to students. This ties directly to improving stress levels as both students and TAs will need to spend less time going over basic code issues that could be solved without the process of office hours or a simple hand raise.

A broader context-principle pair that our project is lacking a little in is economics. While our design has a minimal cost impact, it is rather difficult to measure or predict, given that it is highly dependent on Iowa State University resources. To better accommodate for this, the team will need to investigate maintenance costs more closely from different perspectives, such as how the Electronics and Technologies Group at Iowa State handles maintaining proprietary software and what their protocols for that are.

7.3 Virtues

<u>Three virtues that are important to our team are:</u>

- Cooperativeness: a willingness and ability to work with others
  - Cooperativeness is very important in making sure our team can make progress on the project without getting in each other's way. We will communicate frequently and make it clear what we are currently working on so as to not create confusion.
- Clear and thorough documentation: detailed documentation that avoids unnecessary complexity and covers all aspects of the project.
  - We will document the changes we have made to the previous team's work and provide clear instructions on how to use the critiquer.
- Honesty: being truthful and communicating openly
  - As a team of individuals with our own ideas and opinions, we will continue to openly express them. If someone disagrees with a decision an individual has made, they will bring their thoughts forth, not simply sit back and accept the decision without any discourse.

<u>Individual Reflections:</u>

- James Joseph
    - One important virtue that I feel I've demonstrated best is communication. With our team, I feel like I've been able to express my ideas and thoughts around the direction of the project, and I've been able to listen to other people's thoughts about direction. While the more cooperative nature of our communication has led to more overhead, I believe that it has been worth it to bring out the best idea of what the project should be
    - The most important virtue to me is work competence. I feel that projects are nothing without a good result. My biggest effort will be towards making the product functional and worth the user's time. Even with this, I have felt that my performance this semester has been a little lackluster as I've waffled too much with the direction and task breakdowns to tackle. Coming into the next semester, I will lay out a more definitive direction and focus on developing and completing a backlog.
- Samuel Lickteig
    - A virtue I feel I have demonstrated thus far is responsiveness. Responding quickly to team members helps their work flow more smoothly. I try my best to respond to the Discord as quickly as possible so as to not potentially delay their work.
    - A virtue I need to work on is decisiveness. It is important to me because there are many small decisions that need to be made when working on a large project like this. Spending too much time questioning my own thoughts results in me putting more time than necessary into what should be small decisions. I will try to demonstrate this by having a more structured means of comparing options that results in a quicker resolution.
- Alix Noble
    - A virtue I feel I have demonstrated well is civic-mindedness. Always keeping the end users and our clients in mind is important to the project's success, as they will ultimately be the ones who use the system. I feel like I always keep in mind how decisions will impact the user experience. Whenever I am making a decision about something in the project, I am cognizant of how that will ripple into the usability and social usage of the tool.
    - A virtue I want to demonstrate more of is clear and thorough

documentation. A lot of the project is lacking documentation or code comments about how different parts of it work. Demonstrating this virtue more by writing documentation about the things I work on and commenting code will go a long way toward our goal of modularity and ease of maintenance.

- Andrew Sand
    - A virtue I feel I have demonstrated in my work throughout the project thus far is honesty. I do my best to be as honest as possible when communicating with others and doing work for this project, as it ensures that everyone is in agreement regarding a topic, concept, or idea. I am never intentionally dishonest about information or work that needs to be communicated. Honesty is extremely important for this project due to the number of parties involved with its creation and who will eventually use the finished version of it in the future. It is critical to be honest and truthful with what the project can and can not do, and each party needs to trust each other for this to be successful.
    - A virtue I wish to work on in the upcoming second half of this project's development is my willingness for self-sacrifice. Throughout the project's first phase, I was not particularly happy with the amount of time I dedicated to working on it, and I feel that there needs to be more on my end to match the other member's progress. There must be a delicate balance of willingness for self-sacrifice on the project, not too much but not too little. In the future, I wish to dedicate more of my time toward working on the project to not let the team down and ensure that I am putting in my fair amount of effort to complete the project.
- Owen Sauser
    - The virtue that I have demonstrated is the virtue of commitment to the public good. I have always made preventing cheating with our product a priority. Doing things like suggesting a way to disable the tool during exams or logging the submissions. Then also we want the students to use our product to supplement their learning, not to replace it.
    - The virtue that is important to me that I have not fully demonstrated thus far is timeliness. This is because I had trouble keeping up with the deadlines we had set. While some of this is due to factors I am not able to control, there were other times when it was definitely a lack of motivation on my part. And in the upcoming semester I will be working to prevent this in the future by setting aside more time for the project

8.0 Conclusions

8.1 Summary of Progress

The sdmay25-23 team has accomplished much during the year of development that was allotted to the Code Critiquer System for the C Language and Embedded C. At the beginning of the work period, the team was handed a prototype of a static analysis tool for the C programming language and the concept of transforming it into something that could be used as a supplemental tool for CPR E 2880. In this regard, after meeting extensively with the project's advisor and client in addition to several different involved parties, the team has created a final deliverable that can be used to analyse C programs with an emphasis on CPR E 2880 tools, standards, and concepts while also giving instructors and teaching assistants the information they need to identify problem subject and enhance their course.

Since the start of the project, the team has built off of the sdmay24-34's prototype by getting it running, overhauling the user interface, implementing more than just instructor accounts, overhauling the static analysis pipeline, implementing an analytics page, improving account security, fixing bugs, integrating optional Code Composer Studio compilation, implementing student feedback, reworked the documentation, and tons of other smaller changes to greatly improve the user experience. A big part of the team's success can be attributed to constant communication between members but also with the project advisor, and other third parties related to the CPR E 2880 Iowa State course. An unwavering focus on user requirements was the driving force of the design and task prioritization, leading to a satisfactory final deliverable that can be prototyped as is and further expanded upon by future groups.

8.2 Value Provided

Overall, the final design does a great job at addressing user needs. The team ensured to keep user requirements and needs at the forefront of the design by constantly evaluating different use cases, frequently meeting with the project's advisor and client who is a CPR E 2880 instructor, and consulting with teaching assistants and students who have taken the course. With what the team has accomplished with the project, there is now a competent system that can have a trial run in a future CPR E 2880 class. This prototype will allow students, professors, and teaching assistants to test the system first hand in class to get feedback, discover issues that need resolving, and further refine the functionality that the critiquer should provide. There are quite a few opportunities for future design team endeavors regarding the work completed thus far, and several possibilities are described in the next section.

8.3 Next Steps

While a significant amount of progress has been made on the code critiquer system with a solid foundation being established, there are still many ways that it can be improved or expanded upon. When brainstorming and defining the scope of the project with the client and advisor, there were numerous ideas that were discarded or designated as stretch goals not because they were bad or unimportant to the system, but due to time and knowledge constraints.

The most prevalent of these cut functionalities is implementing dynamic analysis into the system. Given CPR E 2880’s limited lab times and the even more restricted teaching assistant availability, it would be immensely beneficial if students could test their code in a CyBot environment while being off campus or outside of lab times. With dynamic analysis, a student would upload their code to the system, turn on the optional dynamic checking, and have the server run their code before reporting back on how the program performed on a CyBot. Given the unique logistics of this, the team discussed several ideas about how to tackle this concept, especially since it was originally part of sdmay24-34’s design before being scrapped in favor of a more robust scope (See Appendix 2 for more details). One of these ideas was that there was a CyBot emulator developed at Iowa State during the 2020 pandemic, which might prove a useful starting point for implementation into the code critiquer. It allows for students to run code in a simulated CyBot environment as hoped, but it still requires a connection, albeit remote, to a physical TIVA board.

Additionally, there are a number of improvements or minor functionality additions that can be made to improve the overall usability of the code critiquer. Some of these include further user interface improvements, creating a mobile-friendly version of the frontend, improving frontend accessibility to be more easily read by screen readers, and improving the instructor analytics page. A larger issue that should also be tackled some time in the future is the ease of adding further antipatterns into the system. Presently, antipatterns that should be recognized by the critiquer either need to be hardcoded into the server or implemented using regular expressions in the instructor page. Essentially, there is presently a high barrier to entry for adding or editing antipatterns, which should be lowered, allowing for easier tailoring of the system. The team has also considered adding Microsoft Authentication Library (MSAL) support, which would make adding students, teaching assistants, or instructors to the system significantly easier. With MSAL implemented, the system would be able to connect with Iowa State's Microsoft accounts and Active Directory, automatically retrieving user information and roles without having to manually upload a file of a specific format to accomplish this.

Generalization was also a highly discussed feature for the code critiquer. The Michigan Tech University static analysis tools that this project is inspired by have different versions for various programming languages. Given that this code critiquer project has a lot more functionality regarding using it as a tool in university classes, it would be extremely beneficial if multiple languages could be supported so that is could be used in other courses at Iowa State, such as SE 1850, COM S 2270, or COM S 2280. While the team focused on making aspects of the system generalized and kept that idea close to the center of the team's designs, there is still a sizable amount of work that could be done to make this a more general tool for different environments, classes, and languages.

Finally, there have also been discussions with the project's client and advisor regarding the usage of artificial intelligence and large language models with the code critiquer. Several meetings toward the beginning of the project revolved around the reuse of sdmay24-34's code or if it should be rewritten with something like AI in mind. Ultimately, it was decided to focus on static and dynamic analysis that is hardcoded, but the idea was never fully dismissed. The reasoning behind not implementing it was a mixture of no one on sdmay25-23 having much experience or interest with those technologies, concerns over the correctness of AI leading to confusing students further, ethical concerns, and concerns over resource logistics and usage, given that AI would require more computing power than what the project currently uses. However, AI would be a good consideration for future additions, with a potential usage of it being trained on the CyBot and TIVA board manual so that students can query it to easily find specific answers.

9.0 References

[1] *Code Critiquer System for the C Language*. [Online]. Ames, IA: Iowa State University, 2024. Available: https://sdmay24-34.sd.ece.iastate.edu/.

[2] Albrant, L., Pendse, P., Dasker, D., Brown, L., Sticklen, J., Jarvie-Eggart, M. E., & Ureel, L. C. (2024). Work-in-Progress: Python Code Critiquer, a Machine Learning Approach. Proceedings - Frontiers in Education Conference, FIE. http://doi.org/10.1109/FIE58773.2023.10343017.

[3] *PC-lint Plus 2.2*. [Online]. Stuttgart, Germany: Vector Informatik GmbH, 2024. Available: https://pclintplus.com/.

[4] *CodePal C Code Reviewer*. [Online]. Nigeria: CodePal, 2024. Available: https://codepal.ai/code-reviewer/c.

[5] 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.

[6] 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.

[7] *Code Composer Studio*. [Online]. Dallas, TX: Texas Instruments Incorporated, 2025. Available: https://www.ti.com/tool/CCSTUDIO.

10.0 Appendices

Appendix 1 - Operation Manual

Create .env File

1. Open a python shell
2. Run `import secrets` and `secrets.token_hex()`
3. Create a `.env` file in the root directory of the project
4. Enter the following into the file:

```
ENVIRONMENT="qa"
FLASK_SECRET_KEY=[secrets.token_hex() result here]
FLASK_DATABASE=sqlite
LIBCLANG_LIBRARY_PATH=./library
ECLIPSEC_LOCATION="Path to Eclipsec.exe or Eclipse on Linux inside of Code Composer Studio installation"
WORKSPACE_DIR="Path to CCS folder in project"
TIVA_LOCATION="Path to TivaWare_C_Series-2.1.4.178 library directory"
```

How to Run

1. Create `.env` file if you haven't already
2. Open terminal in project directory
3. Install the requirements: `python -m pip install -r requirements.txt`
4. Initialize the database: `python -m flask --app . init-db`
5. Install Docker
    a. Windows: Install Docker Desktop and make sure it is running while the code runs
    b. Linux: Install docker with your package manager (Eg. `sudo apt install docker`)
6. Install Code Composer Studio Version 12.7.1
    a. https://www.ti.com/tool/download/CCSTUDIO/12.7.1
7. Install TivaWare_C_Series-2.1.4.178 drivers
    a. https://www.ti.com/tool/download/SW-TM4C/2.1.4.178
    b. If installing on linux you may need to extract the file contents from the exe
8. Start the application: `python -m flask --app . run --port 8000 --debug`

9. Open localhost:8000 in your browser

Accessing the database in the code

1. Import database: from database import get_db
2. Example for fetching data:

```
db = get_db()
results = db.execute(
    "SELECT long_desc, short_desc FROM pattern"
).fetchall()
print(results[0]['long_desc'])
```

3. Example for changing the database:

```
db = get_db()
db.execute(
    "INSERT INTO pattern (long_desc, short_desc) VALUES (?, ?)",
    ("long description", "short description"),
)
db.commit()
```

Testing / Running critiquer files that access the database without Flask

1. Add this to the top of your test file: from test import app
2. Before calling the function to test, put the call in this with statement: with app.app_context():

Appendix 2 - Alternative/Initial Version of Design

Before proper development began on the code critiquer system, there was an initial design that the team formulated, which was reevaluated during development and transformed into the final design. This original design focused less on the user functionality and Code Composer Studio side of things, instead implementing dynamic analysis using either a remote connection to a TIVA board or a simulated one. Either option would have allowed students to upload their code and have it run in a real or simulated CyBot environment, and the critiquer would give feedback based on the running behavior of the student's program.

Additionally, the team had original plans in place for expanding the language capabilities of the system, allowing it to be more generally used in other Iowa State courses or with other programming languages such as Python or Java. This version of the system would have been closer to what Michigan Tech University, which is what this project is inspired by, accomplished with their static analysis tools. However, after continued conversations with the project's advisor and client, the scope was refined and made less generalized, opting to focus more on targeting the CPR E 2880 course work and topics given the available time of the team members.

Reasons for Change:

- Dynamic analysis is significantly more complex than static analysis (High risk)
- Reevaluated scope and goals from the client and advisor
- All parties would rather have a more functional but nicher system
- Time constraints
- Additionally functionality and generalization can be implemented in the future by other teams, but a solid core foundation needed to be established first

Appendix 3 - Other Considerations

Throughout working on the project, each team member has committed a lot of work to the system that they are proud of accomplishing. From contributing hours weekly, meeting twice per week, writing documentation, and presenting the work completed, team sdmay25-23 is pleased with the progress made, knowledge gained, and final deliverables.

Overall, many of the team members learned:

- New technologies (Flask, Python, Docker, and Bootstrap)
- New programming concepts (Regular Expressions, Abstract Syntax Trees, and Static Analysis Methods)
- How to work on a larger scale project compared to average course work (Closer to an industry scale project)
- How to gather and construct a design around user requirements with many parties involved

Appendix 4 - Code

Code for the project can be found in the team's GitLab repository:
https://git.ece.iastate.edu/sd/sdmay25-23

Appendix 5 - Team Contract

Team Members

- James Joseph
- Samuel Lickteig
- Alix Noble
- Andrew Sand
- Owen Sauser

Required Skill Sets for the Project

- Knowledge of the C programming language - Required for CPR E 2880
- Knowledge of embedded programming - Required for CPR E 2880
- Frontend Web Development - Required for the frontend web application
- Backend Database Development - Required for the backend database and critiquer
- Data Security and Best Storage Methods - Required for storing user data in an appropriate way
- Knowledge of different representations of code - Required for finding antipatterns in code
- Antipattern checking mechanisms - Required for finding antipatterns in code

Skill Sets Covered by the Team

- Knowledge of the C programming language - Andrew, Owen, Samuel
- Knowledge of embedded programming - James, Owen, Samuel
- Frontend Web Development - Alix, Andrew, Samuel
- Backend Database Development - Alix, Samuel
- Data Security and Best Storage Methods - James, Owen
- Knowledge of different representations of code - James, Owen
- Antipattern checking mechanisms - Andrew, James, Samuel

Project Management Style Adopted by the Team

The team adopted a hybrid management style between Waterfall and Agile for this project, fitting the specific project, team, and work schedule. The overarching project is split into Waterfall-like tasks, each needing to be completed before the next, but these tasks are worked on in an Agile-like sprint. Weekly, the team meets to share progress, discuss the project's direction, and plan for the next "Sprint" (week). The team's hybrid approach also draws inspiration from Waterfall's larger tasks that must be sequentially completed before the next can be worked on, as seen in the project's task decomposition chart (See Fig. 1). This combination of the two project management styles has served the team well, striking a balance between responsiveness and a structured work order that suits this project.

Initial Project Management Roles

- James Joseph - Secure System Design, CPR E 2880 Liaison
- Samuel Lickteig - Backend System Design
- Alix Noble - Testing
- Andrew Sand - Team Organization, CPR E 2880 Liaison
- Owen Sauser - Client Interaction, Frontend System Design, CPR E 2880 Liaison

Team Contract

Team Members:

1) James Joseph	2) Samuel Lickteig
3) Alix Noble	4) Andrew Sand
5) Owen Sauser	6)
7)	8)

Team Procedures

1. Day, time, and location (face-to-face or virtual) for regular team meetings:
   Monday 2:00–3:00 PM, virtual through Discord
2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):

    a. Discord group chat for member-member communication in addition to updates, reminders, scheduling, etc.
    b. Email for member-advisor communication
3. Decision-making policy (e.g., consensus, majority vote):

    Three or more members must be in complete agreement with the decision before moving forward.
4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):

    In our Google Drive, which is shared with all group members, there will be a folder that contains summaries of each meeting. This archive will always be available to all members. At least one member present should contribute to the summary. The contributing members will vary based on meeting attendance.

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:
    a. Inability to participate in a meeting or tardiness of more than 15 minutes must be communicated at least an hour before the meeting's scheduled start time.
    b. Members are expected to attend and participate in all advisor meetings unless communicated beforehand.
    c. Members are expected to attend and participate in all weekly meetings unless communicated beforehand.
    d. Additional meetings and their expectations will be discussed on a case-by-case basis.
2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:

    Each team member is responsible for setting and meeting deadlines on the tasks they are responsible for. If a task requires or relies on multiple team members, a discussion will be held to determine an appropriate timeline for each party where the parties will be responsible for their given assignment and deadline.
3. Expected level of communication with other team members:

    Every member of the team must give weekly progress reports that include what was accomplished along with what there is still to do. Any tasks blocking theirs must be communicated immediately in the appropriate communication channel.

4. Expected level of commitment to team decisions and tasks:

   Each team member is expected to be able to complete their tasks in a reasonable amount of time. If they need more time they should communicate what issues they are experiencing with the rest of the team so we can work together to remedy the situation.

Leadership

1. Leadership roles for each team member:
   a. James Joseph: Secure System Design
   b. Samuel Lickteig: Backend System Design
   c. Alix Noble: Testing
   d. Andrew Sand: Team Organization
   e. Owen Sauser: Client Interaction, Frontend System Design
2. Strategies for supporting and guiding the work of all team members:

   As stated in participation expectations, team meetings will include progress reports that allow us to see each others' progress and help if there are questions or concerns.
3. Strategies for recognizing the contributions of all team members:
   a. Git commits and merge requests
   b. Progress reports
   c. Meeting summaries

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.
   a. Alix Noble: Experienced with front end and back end development and with unit testing in Java and JavaScript. Some experience with database management and web development.
   b. Andrew Sand: Experienced with C and low-level development. Additionally is knowledgeable and has experience with frontend web development. Has experience maintaining documentation for development teams.

   c. James Joseph: Experienced with both web development and cybersecurity. Can bring a security perspective along with their experience in programming to develop the application in a way that will keep users and the university protected. On top of that, they have experience with reverse engineering and low-level programming, so they will have a good understanding of what a program aims to do.
   d. Owen Sauser: Experienced with Java, C, C#, and VBA. Has experience in IT and cybersecurity. His current job is translating a company's software from VBA to C#, so he has experience working with other peoples' code and making modifications if necessary.
   e. Samuel Lickteig: Experienced with C code and web development, as well as common mistakes a beginner may make. Also have experience in backend development.
2. Strategies for encouraging and supporting contributions and ideas from all team members:

   Make sure each team member has a chance to voice their opinions on each issue, even if a majority already believe we should address it one way.
3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)

   They can send a message in the Discord group chat, or if they do not feel comfortable with that they can go through the senior design instructors or our advisor if needed.

Goal-Setting, Planning, and Execution

1. Team goals for this semester:
   a. Improve upon the initial design and gain a better understanding of the C programming language along with its patterns/anti-patterns.
   b. Improve the security of the initial design.
   c. Create a final product that can be beneficially used in its intended environment.
2. Strategies for planning and assigning individual and team work:
   a. As a group we will try to assign equal amounts of work to each team member such that each of their tasks is best suited for their skillset.

   b. If any task is larger than the others, this should be communicated and the timeline can be adjusted accordingly.
3. <u>Strategies for keeping on task:</u>
   Each team member will report on their progress at the weekly team meeting. If a team member is concerned about keeping up with their work, they can bring it up at a team meeting and the team can find a strategy to move forward.

Consequences for Not Adhering to Team Contract

1. <u>How will you handle infractions of any of the obligations of this team contract?</u>
   Collectively talk to the member who made the infraction to see why the infraction was made and discuss what can be done to prevent further infractions.
2. <u>What will your team do if the infractions continue?</u>
   Communicate with the senior design course instructors and/or our project advisor to determine a course of action and what can be done in the situation.

---

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) James Joseph	DATE 9/16/2024
2) Samuel Lickteig	DATE 9/16/2024
3) Alix Noble	DATE 9/16/2024
4) Owen Sauser	DATE 9/16/2024
5) Andrew Sand	DATE 9/16/2024