Code Critiquer in C

DESIGN DOCUMENT

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

Team Members/Roles:
Nicholas Carber
Conner Cook
Brandon Ford
Emily Huisinga
Sage Matt
Cade Robison

sdmay24-34.sd.ece.iastate.edu
https://sdmay24-34.sd.ece.iastate.edu

Revised:
September 10th, 2023
Version 1.0

Executive Summary

Development Standards & Practices Used

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

Summary of Requirements

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

Applicable Courses from Iowa State University Curriculum

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

New Skills/Knowledge acquired that was not taught in courses

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

Table of Contents

List of figures/tables/symbols/definitions (This should be the similar to the project plan)

1 - Team

1.1 Team Members

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

1.2 Required Skill Sets for Your Project

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

(if feasible – tie them to the requirements)

1.3 Skill Sets covered by the Team

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

1.4 Project Management Style Adopted by the team

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

1.5 Initial Project Management Roles

Design Document Maintainer / Official Submitter - Emily Huisinga

Website Maintainer - Sage Matt

Main Contact with Dr. Rover - Nicholas Carber

Michigan Tech Liaisons - Brandon Ford, Cade Robinson

System Sketch Maintainer - Nicholas Carber, Conner Cook

Developers - Everyone, to be divided further depending on specifics

2 - Introduction

2.1 Problem Statement

Novice C programmers, and more specifically CPR E 288 students, need an easy to use tool to provide feedback on their C code and help them debug errors in their code.

2.2 Requirements & Constraints

UI Requirements

- The program should be able to be run through the command line and eventually a GUI
- GUI should be simplistic and easy to use for novice programmers
- All feedback for errors should be presented to the users in a way novice programmers can understand

Maintainability Requirements

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

Functional Requirements

- Files in C should be uploaded successfully
- The program should be able to compile and run the uploaded code
- Should catch all compile time error
- Should catch all runtime errors that unexpectedly stop program or prevent program from stopping
- Should catch most style errors
- The program should be able to communicate with a database that stores the C antipatterns

Testing Requirements

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

Legal Requirements

- Ensure that there are no legal blocks with regards to using students' code to test the software

Performance Requirements

- The code analysis process should take a reasonable amount of time
- The file upload should take a reasonable amount of time and have no loss of data

Constraints

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

2.3 Engineering Standards

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

2.4 Intended Users and Uses

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

3 - Project Plan

3.1 Task Decomposition

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

- Connect application to database to read from anti-pattern data
    - Description: The front-end application must be connected to the backend database.
    - Justification: This fulfills the functional requirement that the program should be able to communicate with a database that stores the C antipatterns.
    - Subtasks:
        - Set up toy database to get the initial connection
        - Set up to official anti-pattern database

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

- System identifies desired compile errors
    - Description: The application is able to identify at least 5 compile time errors upon being run with a sample of code.
    - Justification: This is necessary for the functional requirement of catching all compile errors.
    - Sub tasks:
        - Groups of similar anti-patterns

- Provide feedback for at least 1 anti-pattern
    - Description: The application must provide the user with useful information on what their error is and potential ways to fix the error.
    - Justification: This fulfills the UI requirement for providing feedback to the user in a way they can understand and the functional requirement for the program should be able to compile and run the uploaded code.
    - Sub tasks
        - Terminal feedback
        - GUI feedback

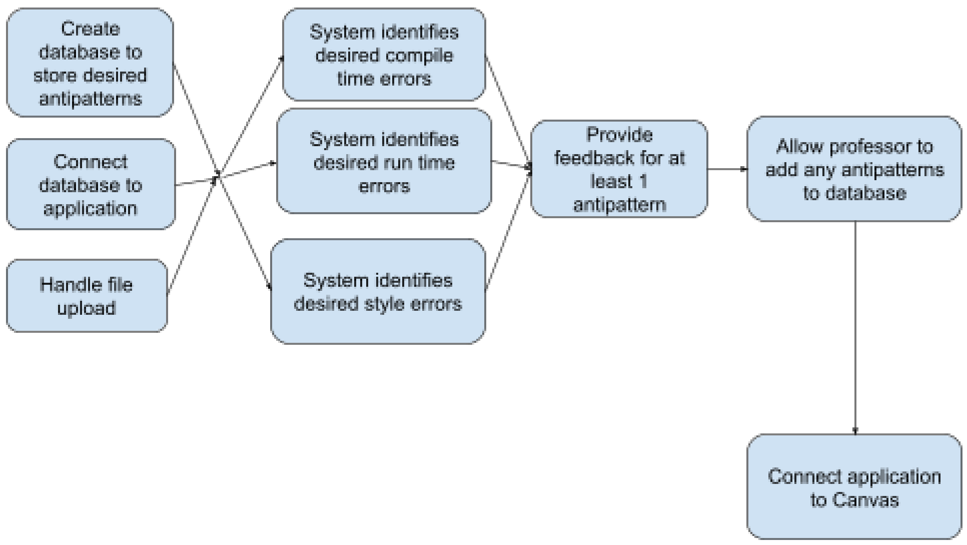
- System identifies desired run-time errors
    - Description: The application is able to identify at least 5 run-time errors upon being run with a sample of code.
    - Justification: This is necessary for the functional requirement of catching all run-time errors that unexpectedly stop the program.
    - Sub tasks
        - Groups of similar anti-patterns

- System identifies desired style errors
    - Description: The application is able to identify at least 5 style errors.
    - Justification: This is necessary for the functional requirement of catching most style errors.
    - Sub tasks
        - Groups of similar anti-patterns

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

- Connect application to Canvas
    - Description: Students should be able to upload their code to Canvas using an external app.
    - Justification: This ties into the maintainability requirement that the project should be of the same design and standard that Michigan Tech has made with their other code critiquers. Currently, other code critiquers work by connecting to Canvas with an external app.
    - Sub tasks
        - Use LTI to connect application

Task Dependency Chart


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

3.2 Project Management/Tracking Procedures

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

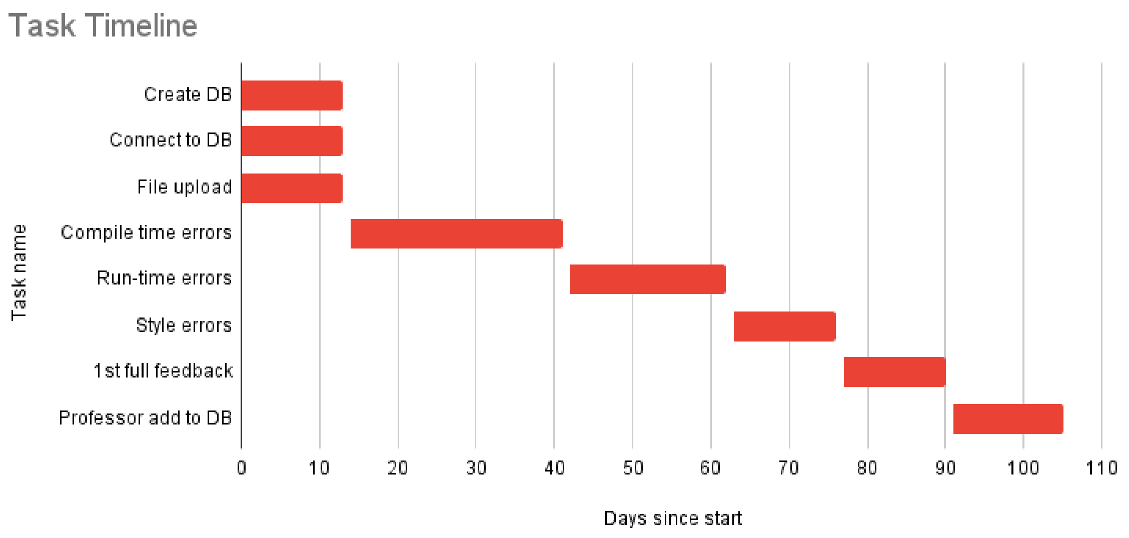
We will use the GitLab repo and the GitLab issues to track our progress and store our project.

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Key milestones:

- Must recognize at least 10 anti-patterns and give valuable feedback
- Should recognize and give feedback on roughly 50% compile-time errors
- Should recognize and give feedback on roughly 50% runtime errors
- Critiquer does not crash on unrecognized errors

3.4 Project Timeline/Schedule



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

3.5 Risks And Risk Management/Mitigation

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

Risks:

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

3.6 Personnel Effort Requirements

Task	**Effort Points**	**Explanation**
Create database for anti-patterns		
Create tables for database	2	Only need 1 or 2 simple tables
Insert anti-patterns into database	20	Need to create regular expressions from anti patterns
Connect application to database		
Create toy database for initial connection	2	Simple initial connection
Connect actual database	3	Experience from initial connection will make this easy

Handle file uploads		
Prompt user for file	2	Command line prompt - “Enter filename:”
Read file data into application	4	Only loading a file into the application and making sure we can access the data after building an Abstract Syntax Tree
System identifies errors		
Compile errors	30	Need to worry about multiple files. Need to run the program through regex data.
Run-time errors	40	Textual analysis done without actually running the program
Style errors	15	Match the regexes from the database
Provide feedback for antipatterns		
Terminal feedback	5	Output formatted feedback generated by the program
GUI feedback	15	Have to build a GUI that will eventually connect to the main application
Allow professors to add antipatterns to the database		
Create interface to add antipatterns	5	Basic form for new antipattern information, which then gets stored in the database
Connect interface to database	5	Create an SQL INSERT statement to put the new anti-pattern data into the database
Connect application to Canvas		
Connect application	20	After completing research, it should be as simple as importing our already completed application.

3.7 Other Resource Requirement

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

4 Design

4.1 Design Content

The Code Critiquer will be first and foremost an application that students can upload their work to, with the ability to configure what errors they want to look for. They would receive feedback that is similar to compiler feedback, except more specific and worded in a way that is comprehensible by even programming novices. Eventually, the application would be hooked up to Canvas as an External App using LTI tools.

4.2 Design Complexity

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

One challenging requirement that we have would be to generate valuable feedback for errors in C code. Currently the GNU C compiler doesn't provide feedback on how to fix certain errors. Another challenging requirement is to allow professors to upload tests to run student's code with.

4.3 Modern Engineering Tools

For the development of this design, we will use figma to create a front-end layout. Depending on the preference of the developer, we will use either Visual Studio Code or JetBrains software for the actual development of the code.

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

4.4 Design Context

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

Area	**Description**	**Effects of Code Critiquer in C**
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)	This project will greatly improve the mental health of students who are struggling to learn programming but feel that they have nowhere to go for help to learn.
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.	There are a number of ways that Code Critiquer in C can both positively and negatively impact our profession. It can be used to create a generation of programmers who have incredibly strong programming skills. On the other hand, if the Code Critiquer in C were to become good enough, future programmers may become lazy and rely on the critiquer to make corrections. It would be similar to how our generation has become increasingly poor spellers as we have become reliant on autocorrect.
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.	This project will have a minimal environmental impact, except for the energy it takes to run the device that the application will be run on.
Economic	What economic impact might your project have? This can include the financial viability of your product within your team or company, cost to consumers, or broader economic effects on communities, markets, nations, and other groups.	This project has the potential to inflate the market with programmers, as students will feel more supported during the learning process and therefore will be less likely to drop out. Additionally, if the Code Critiquer in C technology could become advanced enough, it could possibly create a decrease in demand for programming tutors.

4.5 Prior Work/Solutions

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

Michigan Tech's Python Code Critiquer:

> *Advantages:* Due to many useful libraries that Python has available, we have decided to create our C Code Critiquer in Python. As such, the Python Code Critiquer, which is written in Python, will have the most similar code structure to ours.
>
> *Shortcomings:* As C is a much lower level language and has the tricky issue of memory management, our C Code critiquer will end up being much more complex than the Python Code Critiquer.

Michigan Tech's Java Code Critiquer:

> *Advantages:* Michigan Tech's Java Code critiquer has the most thorough database of antipatterns and will be of the most use to us, as Java and C do have their similarities (though many, many differences).
>
> *Shortcomings:* The Java Code Critiquer is written in Java, and as we will be writing our Code Critiquer in Python, this code structure will be of no use to us.

Michigan Tech's MATLAB Code Critiquer:

> The MATLAB Code Critiquer does not offer anything unique to us that the Python and Java Code Critiquer don't already offer. Nevertheless, it is still a useful tool in our pockets.

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

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

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

> N/A

4.6 Design Decisions

1. We had to make the decision to allow professors to upload their own code.

2. We had to make the decision to implement as a command-line application before creating a GUI/importing to Canvas

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

4.7 Proposed Design

4.7.1 Design 0 (Initial Design)

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

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

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

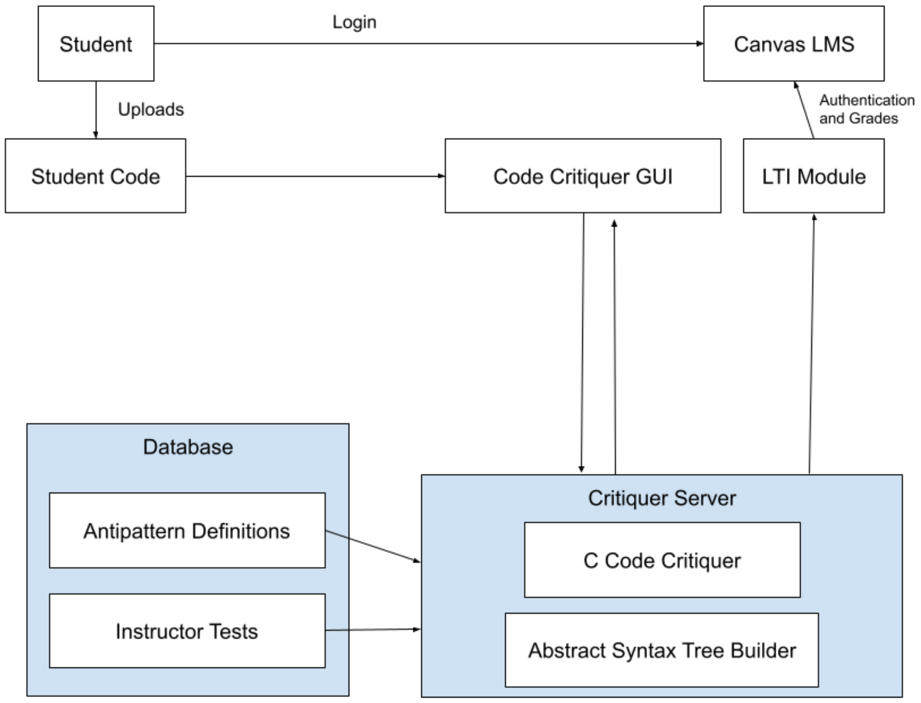
LTI Module: Connection point between Canvas and the Code Critiquer System. The LTI Module follows standard practice for creating the connection between LMS and LTI systems. This will help fulfill the UI Requirement, that the application will have to be run through a GUI (in this case, Canvas).

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

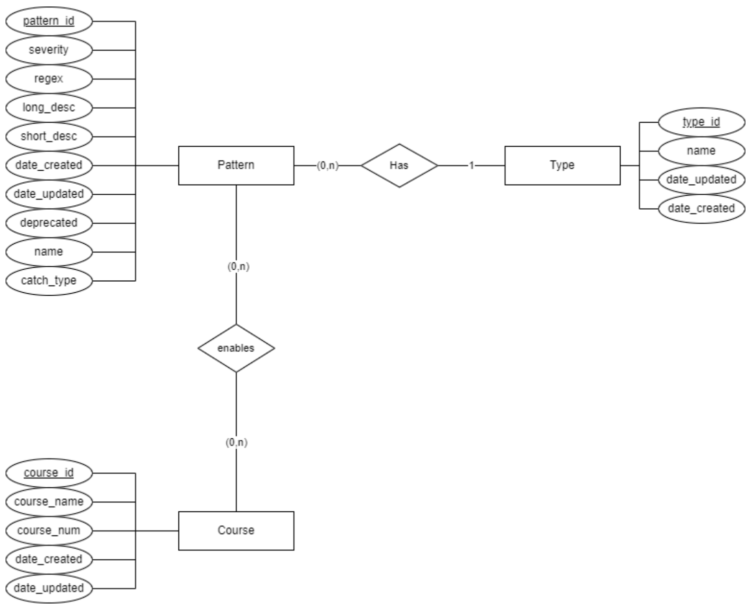
Critiquer Server: This will consist of the code critiquer application and the abstract syntax tree builder. The code critiquer will analyze the student's code by using the anti-pattern data in the database and generate appropriate feedback based on any errors found. The abstract syntax tree builder reads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

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

System Sketch:



ER Diagram of the Database:



Functionality

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

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

4.7.2 Design 1 (Design Iteration)

4.8 Technology Considerations

4.9 Design Analysis

5 Testing

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

5.1 Unit Testing

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

5.2 Interface Testing

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

5.3 Integration Testing

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

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

Finally, we will test the canvas integration with the website. This should be relatively easy as we should be able to just make sure that the website is correctly running inside of the canvas application and is sending data effectively.

5.4 System Testing

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

5.5 Regression Testing

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

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

5.6 Acceptance Testing

- Functional
    - Goals for compile time, runtime, and style errors should be met with appropriate feedback. Specifically, successfully give feedback on at least 5 errors in each category.
    - Critiquer should not extend the time it takes to run a program by more than double. (estimate - can change)
- Non-Functional
    - Command-line and GUI should be intuitive for novice programmers.
    - Modular database with easy extensibility.

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

5.7 Security Testing

For our project, security is not that high of a concern as we are not storing a lot of sensitive information in our database. However, we will still want to prevent any attempts to access the database without permission via SQL injections. Therefore, we should sanitize all of our SQL queries. In the event we do need to store sensitive information, we should hash that information and store the cipher text. Testing these functionalities should be pretty straightforward, as we just need to make sure all queries are sanitized and no plaintext of sensitive information makes it into the database.

5.8 Results

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