**SYLLABUS CHECKER**

**Project Progress Report II**

****

**TEAM**

Jack Demtshuk, Ishaq Halimi, Joscelin Montoya-Rojas, Rebecca Nanayakkara, Nicholas Taweel

**CLIENT**

Dr. Judith Ozment

Division Head of Science and Engineering

[o96@psu.edu](mailto:o96@psu.edu)

Team members responsibilities

* Jack Demtshuk - Developed PDF input and text-extraction functionality; created the keyword/phrase dictionary for sentence-transformer search; refined GUI layout and report output; implemented recommendation text for missing syllabus sections.
* Nicholas Taweel - Implemented spell-checking; developed the initial GUI framework and file-selection workflow; contributed to testing and debugging.
* Rebecca Nanayakkara - Integrated sentence-transformer model for classification; implemented readability evaluation; added interpretive feedback and recommendations; contributed to accuracy testing of the output.
* Joscelin Montoya-Rojas - Conducted testing; assisted with refinement of keyword/phrase dictionary; documented errors, misclassifications, and bugs.
* Ishaq Halimi - Performed additional testing on content-analysis and readability results; supported debugging and helped verify reported issues.

Problem Statement

A syllabus is one of the most important documents a student receives, yet its quality and completeness can vary widely from course to course. Missing grading policies, unclear attendance expectations, or outdated accessibility statements can leave students confused and unprepared. Faculty at Penn State Abington are required to follow the detailed guidelines outlined in the Faculty Handbook, but checking every syllabus against these requirements can be time consuming and inconsistent. Current tools like Grammarly may help with grammar, but they cannot determine whether a syllabus contains all the elements that students and administrators expect.

The Syllabus Checker aims to solve this problem by providing an automated system that evaluates syllabi for both compliance and clarity. Our tool will go beyond grammar checks by identifying missing or outdated content, assessing readability, and generating actionable recommendations for improvement. In doing so, it supports instructors in creating documents that not only meet institutional policies but are also student friendly and accessible.

By automating this process, the project offers benefits across the board. Students will receive clearer, more reliable syllabi that help them plan and succeed in their classes. Faculty will save time and reduce the stress of manual reviews, while administrators can be confident that institutional standards are consistently met. In short, the Syllabus Checker will raise the overall quality of syllabi and improve the academic experience for everyone involved.

Related Work

When it comes to checking documents such as syllabi, there are already some tools out there that can handle parts of the job, but none of them really cover everything we are aiming for. For example, while Grammarly is great at catching grammar mistakes and spelling errors, it doesn’t know what specific content a syllabus is supposed to have. Grammarly won’t flag your syllabus if something important is missing, such as the grading policy, office hours or accessibility statements.

At our university, faculty have access to Appendix E of the Faculty Handbook, which lays out very detailed syllabus guidelines. These policies specify exactly what must be included, such as instructor contact information, office hours, attendance expectations, and more. In addition, according to the Faculty Handbook, copies of all syllabi have to be submitted to the professor’s division head within the first ten days of the semester. This means that reviewing these syllabi can be time consuming and requires a lot of responsibility for each syllabi to be manually checked for the required details.

This is where our project is different. The Syllabus Checker combines the strengths of existing tools while also focusing specifically on what makes a syllabus complete and student friendly. Our tool will not only fix grammar, it will look for required information, flag outdated content, and give recommendations on how to make the syllabus more readable. Ultimately, our tool is designed to support both professors and students, while also saving time for administrators who are responsible for reviewing syllabi.

Features

The “Syllabus Checker” tool we wish to create will scan syllabi for their quality and accuracy. The tool will check for certain required information from a user’s input syllabus, as well as see that it is grammatically correct and comprehensible. The tool would then output a list of sections that it found to be incorrect, a list of sections that it found to be insufficient, among other descriptive flags. Much of this will be detectable by a machine learning model that we hope to create alongside this tool. Things that are not detectable through the machine learning model will be hard coded using binary trees and conditional statements. Syllabi must be in a PDF format, so our tool can scan text directly from the file. If a syllabus were uploaded as an image file, like a .jpg or .png, it would require some form of computer vision and image processing.

Distinct information that the tool will check for include professor’s name, a grading policy, a course description, and required resources and hotlines. Some sections, such as the grading policy, will not be judged on its perceived fairness or quality as that is dependent on the professor and teaching style. Some courses or departments may have their own requirements. A case of this would be a lab course needing to outline lab safety. The naming and structure of each section will not require specific keywords. For example, a section titled “Grading Policy” is as valid as if it were named “How I Grade.” The program will rank syllabi on whether they pass or fail the check, by providing a grading scale where certain aspects that are required must be present or else they automatically fail, and aspects that are highly recommended are not present, or content is insufficient or overly complex, it will pass but with points deducted. When judging these sections, it will check that there is sufficient information. If a particular section is less detailed than it typically is, the tool will flag it. It will also check for the opposite, flagging where

parts are less simple than they should be. It will provide recommendations to the user on how to improve the syllabus based on the sections that fail to reach a passing grade. The tool will also check for syntax and grammar errors.

The tool is meant to be easy to use, with the expected users just being professors. Therefore, the interface will be very simple and only reliant on the inputted syllabus. We hope to make the output clear and concise, highlighting where the syllabus is lacking and recommending changes.

Functional Requirements

* Users can upload PDF syllabi.
* The system extracts and processes text.
* The system identifies required content sections.
* The system calculates readability metrics (FRE, FK Grade, Gunning Fog).
* The system flags missing or insufficient sections.
* The system provides suggestions for improvement.
* The system displays final scoring and analysis in a GUI.
* The system detects filename formatting concerns.
* The system performs basic spell-checking.

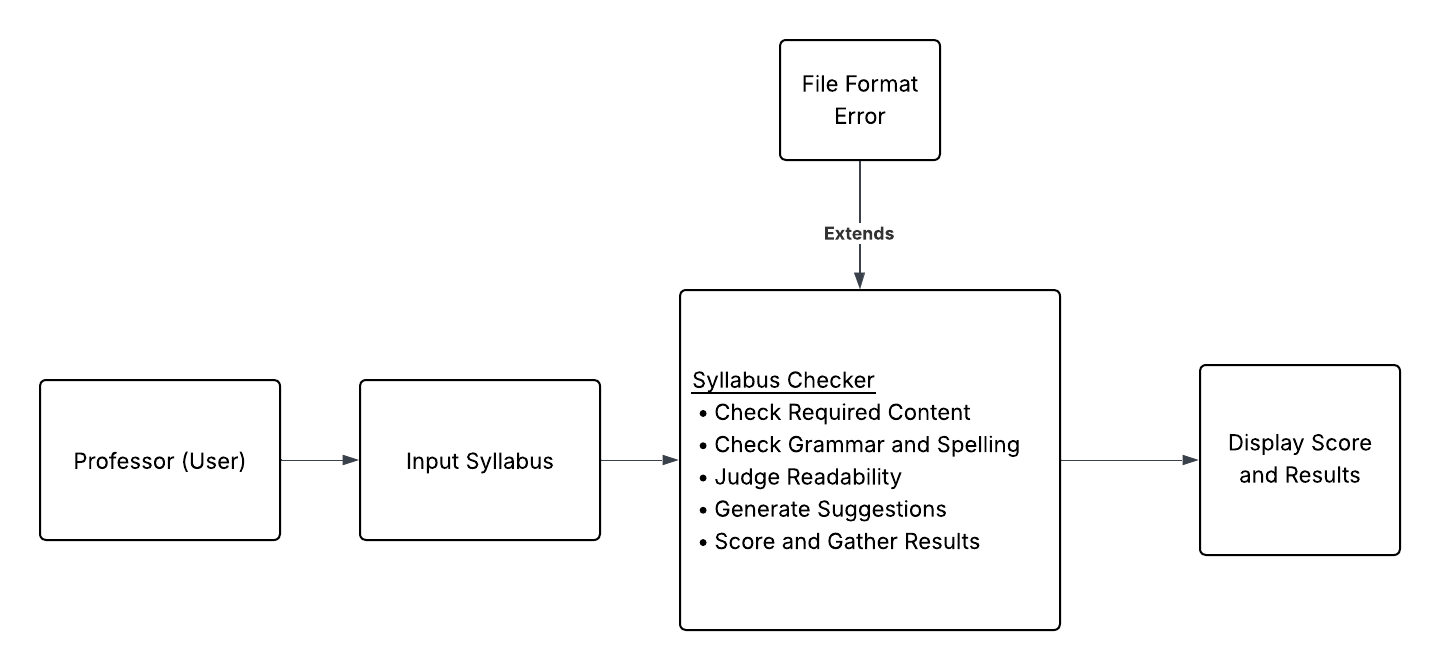
Non-functional Requirements

* Usability: Easy-to-use GUI for non-technical users.
* Performance: Processes typical syllabi in ~10 seconds.
* Portability: Runs locally on Windows/macOS.
* Maintainability: Section libraries/dictionaries should be easy to update.
* Accuracy: Must detect at least 85% of required sections.
* Reliability: Handles irregular or partially scanned PDFs.
* Security: All files remain local to the user’s device.

User stories

* Dr. Judith Ozment: I want to upload a syllabus to verify required content. I want to see which sections are missing. I want readability feedback to simplify confusing text. I want recommendations to make improvements easily.
* As an administrator, I want consistent syllabus quality across courses.

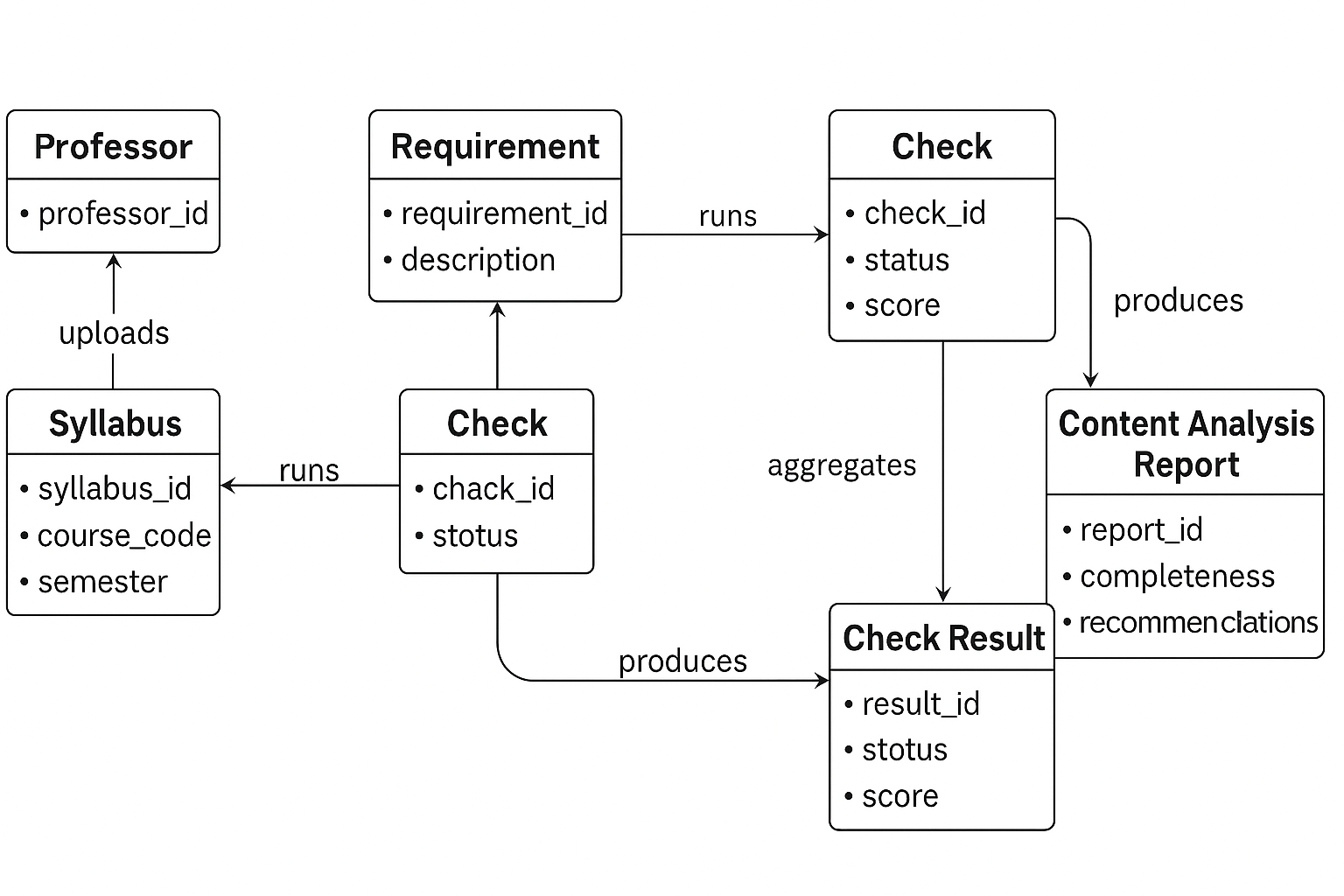
Use Case Diagram

****

**Figure 1. Use Case Diagram**

The use-case situation involves the professor inputting a syllabus into the syllabus checker, where the program then judges the content and quality of the syllabus. A file format error may be raised, which is an extension of the syllabus check step. Given that no error occurs, the syllabus checker will display a score and suggestions.

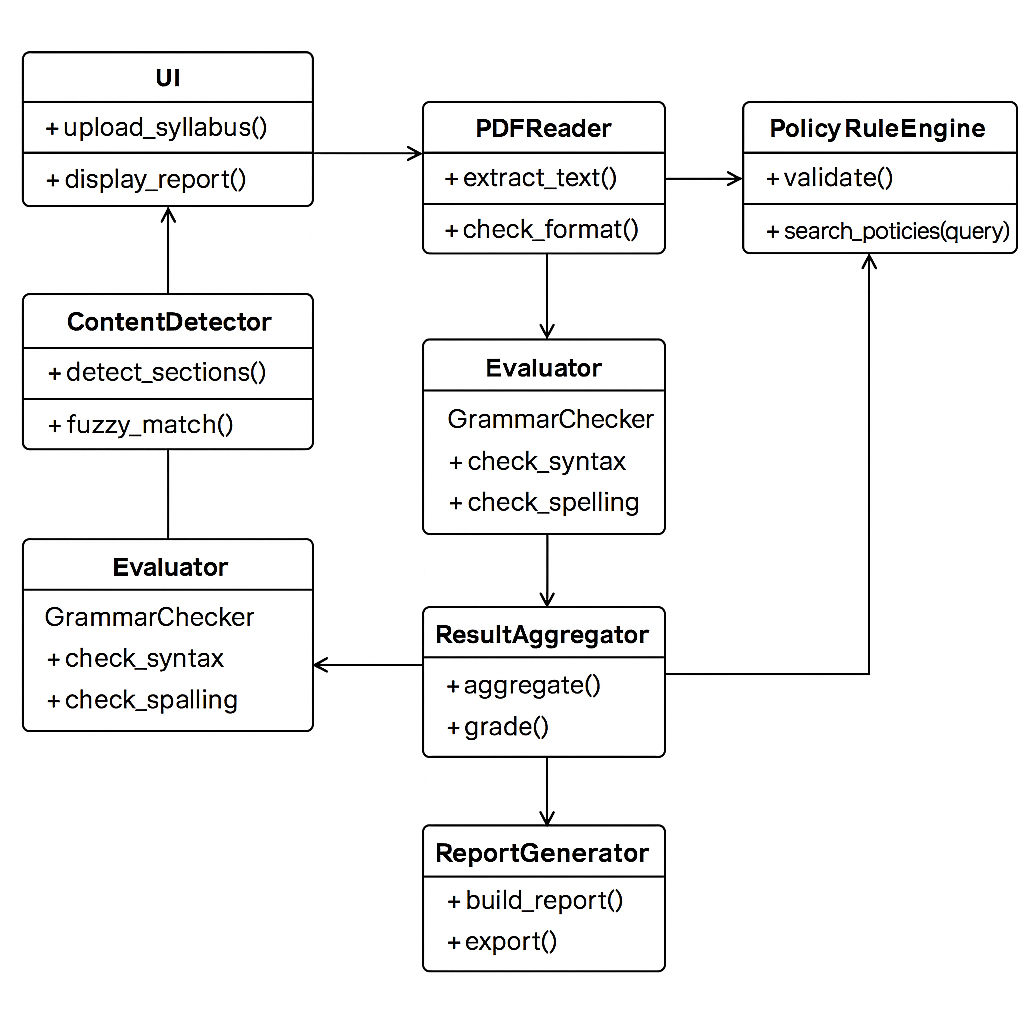
Design Specification



**Figure 2. ER Diagram - Core Data Relationships in the Syllabus Checker**

The ER diagram above illustrates how data flows through the Syllabus Checker system. A Professor uploads a syllabus, which is associated with a specific course. Each syllabus is evaluated against a set of requirements defined by the university’s Faculty Handbook and Appendix E. These requirements generate corresponding checks that analyze the syllabus for completeness and accuracy.

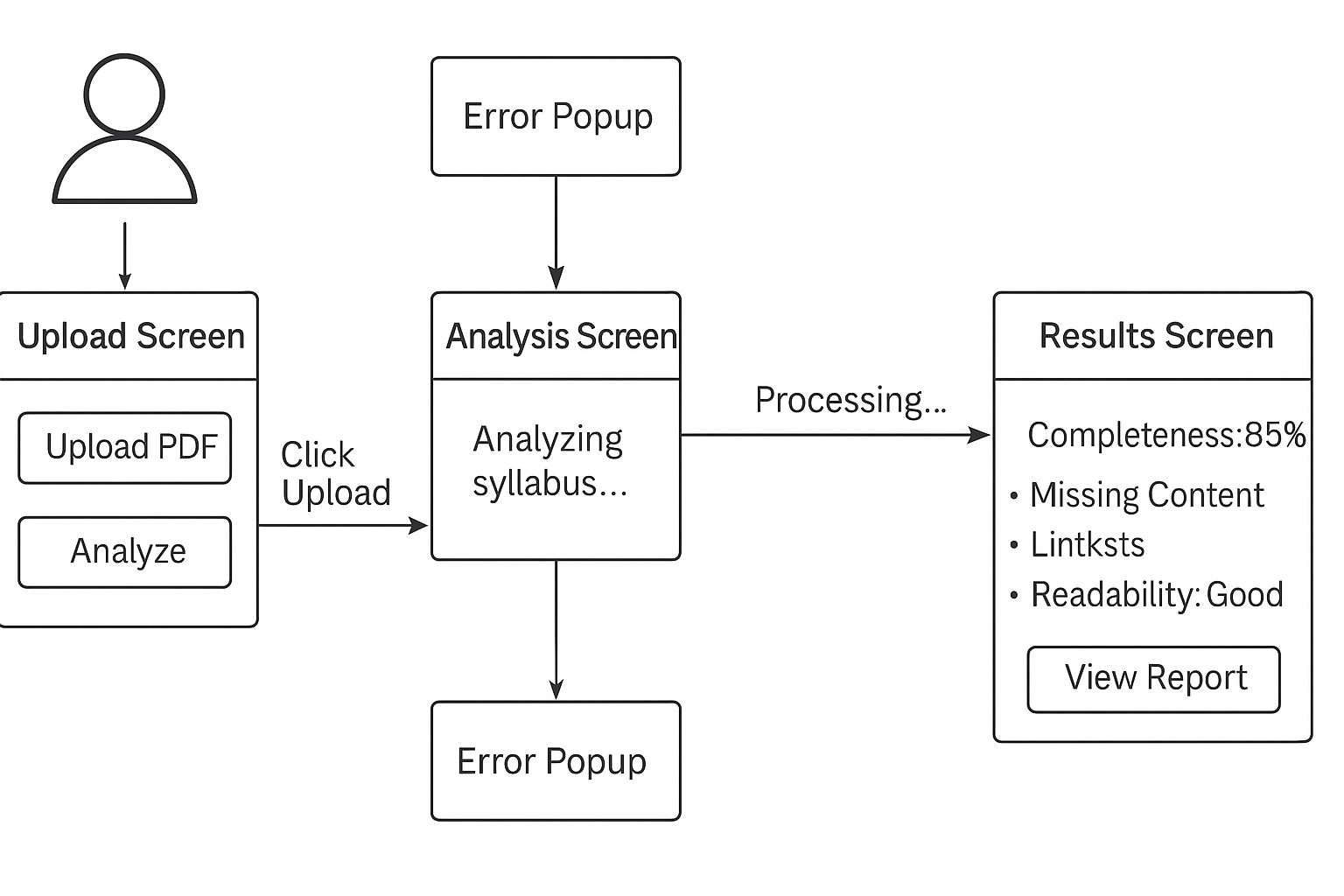
Each Check produces a Check Result, which captures details about whether the required information was found, missing, or outdated. These individual results are then accumulated into a content analysis report, which summarizes the overall syllabus quality, readability, and compliance with institutional guidelines. The requirement entity also informs the final report by providing the context for each evaluation criterion. This structure ensures that every syllabus is assessed systematically and that feedback is accurate, transparent, and consistent across all courses.

****

**Figure 3. UML Class Diagram — Core Program Architecture of the Syllabus Checker**

The UML Class Diagram illustrates the structural design of the Syllabus Checker system and how its classes interact. The UI class enables professors to upload syllabi and view results. The PDFReader class extracts text and validates the file format, while the ContentDetector identifies required sections through fuzzy matching. The PolicyRuleEngine validates content against institutional policies, and the Evaluator (which includes the GrammarChecker) reviews syntax and spelling for clarity.

Outputs from these modules are compiled by the ResultAggregator, which calculates a completeness score and summary data. The ReportGenerator then creates a final content analysis report displayed through the user interface. This modular design supports easy maintenance, scalability, and clear organization of system responsibilities.



**Figure 4. User Interface Diagram — Screen Flow of the Syllabus Checker**

The User Interface Diagram illustrates the user interaction flow for the Syllabus Checker system. The process begins with the upload screen, where the professor selects and uploads a syllabus file. If the uploaded file is not in the correct format (for example, not a PDF), an error popup appears to notify the user. Once a valid file is uploaded, the system transitions to the analysis screen, which displays the progress of the syllabus evaluation, including content verification, grammar and spelling checks, and readability assessment.

After the analysis is complete, the program navigates to the results screen, which presents a comprehensive report including the syllabus completeness score, missing sections, readability score, and actionable recommendations for improvement. The layout is designed with simplicity and clarity in mind, ensuring that professors can easily interpret results and make necessary adjustments to their syllabi.

Required Tools and Technologies:

Our project uses a variety of tools and technologies that work together to analyze and evaluate syllabi efficiently. The system is developed in Python 3.11, which provides the flexibility and libraries needed for text processing, PDF handling, and machine-learning integration. The early prototype, developed using Streamlit, allowed users to upload a syllabus PDF and view simple results in a browser interface. The current version, built with Tkinter and threading, offers a more responsive and stable desktop application that performs the same tasks locally while allowing the analysis to run in the background without freezing the interface.

The program uses PyPDF2 to extract text directly from syllabus PDFs, RapidFuzz for fuzzy keyword matching, and the Sentence-Transformers CrossEncoder model to identify required sections through semantic similarity. This model provides the project’s machine-learning capability, enabling it to recognize important syllabus elements even when they are phrased differently. Textstat is used to calculate readability scores—including Flesch Reading Ease, Flesch-Kincaid Grade Level, and the Gunning Fog Index—which help measure how understandable the syllabus is to students.

In addition, a standalone PySpellChecker module has been implemented to identify possible spelling mistakes, which may be integrated into the main application in future updates. Supporting tools such as NumPy and Python’s built-in regular expressions (re) library assist with text processing and sentence parsing. Together, these technologies allow the Syllabus Checker to provide accurate, automated feedback on completeness, readability, and clarity while maintaining an efficient and user-friendly workflow.

Implementation Details

**Steps/Procedures Required to Develop the Program:**

To build the syllabus analyzer, we decided to split this into two main parts:

1. A backend function that performs all analysis on the PDF
2. A Tkinter GUI class that imports and calls the function

Backend Procedures

1. Import and configure required libraries.
   * PdfReader for extracting text from PDFs
   * os for checking the file path and reading the filename
   * CrossEncoder for semantic scoring
   * re for sentence splitting
   * numpy for converting logits into probabilities
   * textstat for readability metrics.

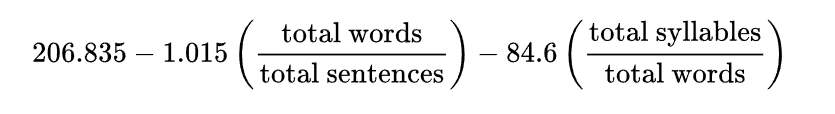
File Validation - Before doing anything, the program checks that the file actually exists and ends in .pdf. If the file is missing or the wrong type, the program will stop and return an error message instead of attempting to analyze it.

PDF text extraction - Using PdfReader, the program loops through each page and calls extract\_text(). All extracted text is stored in a single string. If a page has no readable text (common for scanned pages), the program adds a placeholder like [No Text Found] so the final report reflects exactly what was and wasn’t readable.

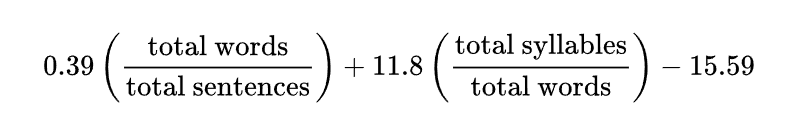
Sentence splitting - The combined text is broken into separate sentences using regular expressions. Extremely short lines or lines with no real words are removed so the next step only runs on useful sentences.

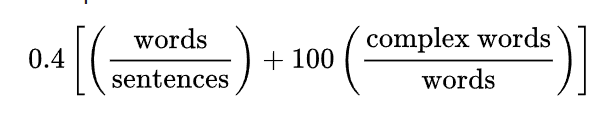
Readability analysis - The program calls textstat to calculate:

* Flesch Reading Ease - Higher scores indicate material that is easier to read, while lower numbers mark passages that are more difficult to read. The formula for this test is:



* Flesch-Kincaid Grade Level - Same concept as flesch reading ease, only this test presents a score as a U.S. grade level. This function is primarily used by teachers and parents to judge the readability of various books and texts. The formula for this test is:



* Gunning Fog Index - This is a readability test for English writing. The resulting index estimates the years of formal education a person needs to understand the text on the first reading. The function determines “complex” words as words consisting of three or more syllables, and does not include proper nouns, familiar jargon, or compound words (such as -es, -ed, or -ing) as a syllable. The formula for this test is:

The results of these tests are then printed in a “READABILITY REPORT”. If the values show that the syllabus is unusually difficult to read, the program applies a small penalty to the final score.

Semantic Detection of Required Syllabus Sections -

After the PDF text is split into sentences, the program loads a pretrained CrossEncoder model from the sentence-transformers library. A CrossEncoder is a type of deep learning model designed for semantic similarity, which means it can determine how closely two pieces of text relate in meaning.

This specific model has already been trained on millions of text pairs from large-scale datasets. During training, the model learned patterns about how real sentences relate to questions, instructions, and statements. Because of this, the model can recognize syllabus information even if the wording is not an exact match.

In this program, each required syllabus section is represented by a short query string like:

“Grade Breakdown: grading policy grade distribution grading scale”

The algorithm works like this:

1. The program takes the query string and pairs it with every sentence extracted from the syllabus.
2. Each pair is fed into the CrossEncoder at the same time (query + sentence).
3. The model outputs a logit score, which is a raw numeric value that represents how strong the semantic similarity is.
4. These logits are converted into probabilities using the sigmoid function, which gives a value between 0 and 1, where numbers closer to 1 mean a stronger match.
5. For each required section, the program looks at the highest-scoring sentence in the entire document:
   1. If the highest probability is above the threshold (currently set as 0.05), the system declares that the syllabus contains the section.
   2. If all probabilities are below that threshold, the section is marked missing.

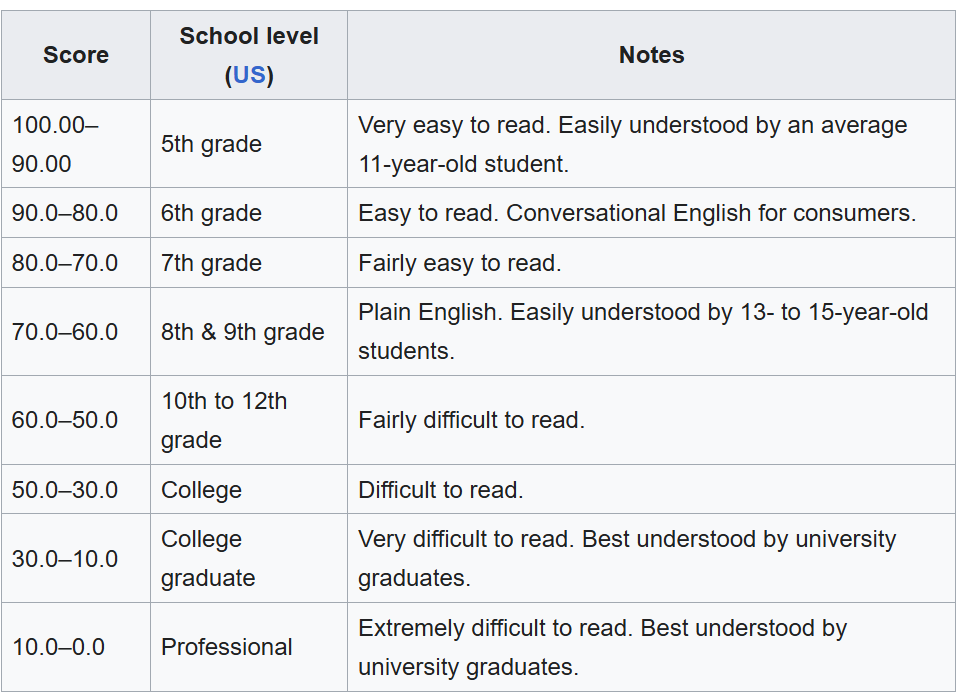
Scoring and Output Formatting - Once every required section has been evaluated, the program calculates a score:

* +10 points for each required section that the model marked as found.
* Readability penalties (-5 or -10) are applied only when the scores show extremely difficult reading levels.

After scoring is complete, the backend generates one formatted text report. The final output includes:

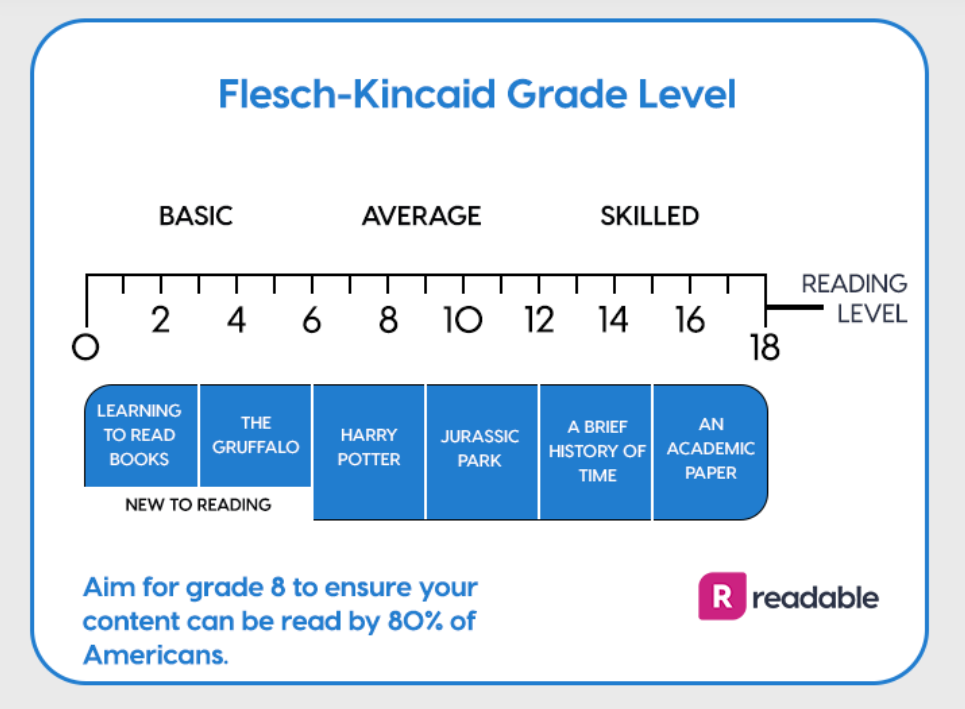
1. Course information extracted from the filename
2. Readability results and explanations of what they mean.

* We used sources to determine our interpretation of the readability scores. For flesch reading ease, Wikipedia provides this scoring table:



Therefore, from the table, we can see that the appropriate score should be in the range of 30-50, where all values above 50 are very easy to read (but no penalty), values between 10-30 are at a college graduate level (-5 penalty), and values under 10 are at a professional level (-10 penalty).

* Readable provides this scoring table for flesch-kincaid grade level:



From this table, we can determine that below 12 should be below a college level, and above 18 would be past college level. Therefore, our scoring should issue a penalty for scores above 18. It should also notify the user that they are in the appropriate level when the score is between 12 and 18, and should notify users that their text is below college level when less than 12, but issue no penalty.

* Wikipedia provides this scoring table for gunning fox index:



From this table, we can see that the ideal score is between 12 and 17, where all scores below 12 are below college level (but no penalty), and all scores above 17 are at a postgraduate level (-10 penalty).

3) A list of found sections with confidence scores

4) A list of found sections

5) Recommendations explaining what the syllabus should include for each missing item

The GUI simply prints this final output to the text box. Since everything is already formatted in the backend, the GUI does not need to process or rearrange any text.

GUI Procedures - The GUI runs the analysis process in the background, so that the GUI can remain operational while this is being performed, to ensure it does not crash when a user starts clicking or scrolling during analysis.

GUI Implementation - A Tkinter window is created with a title, a dark theme, a PDF selection button, an “Analyze Syllabus” button, a scrollable text box, and a status bar at the bottom.

Importing and Calling the Backend Function:

The GUI imports the backend using:

from Syllabus\_Checker\_For\_GUI import check\_syllabus

When the user clicks Analyze, the GUI calls check\_syllabus(self.file\_path) and prints the returned report into the text box.

Maintaining Responsiveness with Threading - The PDF analysis runs inside a threading.Thread. This prevents the GUI from freezing while the CrossEncoder model loads and processes the text.

**Current Status of the Development**

Completed so far:

* The backend analysis function is fully working, only requiring a few small additions.
* PDF validation and text extraction are implemented, including placeholders for unreadable pages.
* Sentence splitting and noise filtering are implemented.
* Readability analysis runs correctly and prints a “READABILITY REPORT”.
* The CrossEncoder model loads and evaluates every required syllabus section.
* Scoring system is implemented (+10 per section, penalties for difficult readability).
* Final formatted report is generated with found/missing sections and recommendations.
* The Tkinter GUI runs, allows PDF selection, and displays the full report.
* Threading is working – GUI does not freeze during analysis.

Incomplete / In Progress:

* Showing example sentences that triggered each “found” section.
* Adding options to copy or save the report
* Inclusion of few additional required sections as per client’s preferences
* Disabling Analyze button while thread runs to avoid end-user error
* Additional testing and threshold tuning
* Additional improvements to the GUI
* Figuring out how to make this program easily accessible for administrators and program heads.

**Revised timeline**

* **Week of September 23: Acquire PSU Guidelines and Recommendations**

We will start by reviewing all syllabus policies and guidelines from the Penn State Abington Faculty Handbook (2025-2026) to understand what is required in a syllabus, as well as look into and review what is recommended to be included in a syllabus by Penn State.

* **Week of September 30: Review Sample Syllabi**

We will then review sample syllabi to identify common features and formatting. This will help us build a flexible template that our program can use as reference. This will include training a machine learning model to identify aspects of syllabi that may be slightly different in presentation, but adequate in content, such as labels like “How I Grade” substituting the ‘official’ “Grading Policy”.

* **Week of October 7, 14: Build the First Version of the Program**

Using the template, we’ll create a program that can scan a syllabus and rate it based on its readability, whether it meets PSU Abington’s requirements, and overall user-friendliness.

**October 16:** Present project updates to the class.

* **Week of October 21, 28: Improve Program**

We have built on our initial program and added many new features, alongside a fully operational GUI. The current state of the program is near completion, only requiring a few more tweaks.

* **Week of November 6, 11: Add Final Improvements and Features**

The last features and improvements that are needed to be added, and any suggestions from our client will be done here. We will also begin testing and refining our program’s search system throughout testing.

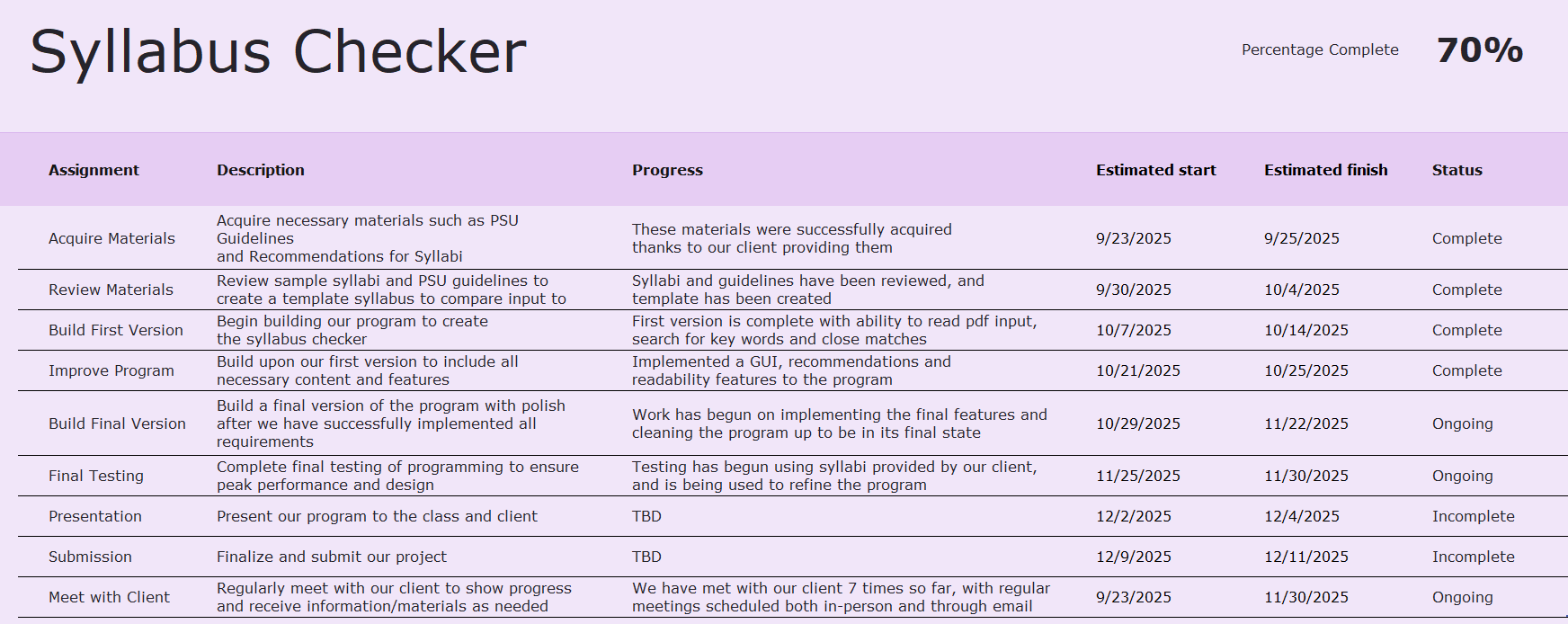
**November 6:** Present project updates to the class.

* **Ongoing: Regular Check-ins with Client**

We plan on meeting with our client, Dr. Judy Ozment, every two to three weeks to ensure that we’re on the right track and that the project meets Dr. Ozment’s expectations.

**Final Testing and Presentation**

* Week of Dec 2, 4: Final Project Presentation
* Week of Dec 9, 11: Submit final project package, including:
  + Final Project Report (PDF)
  + PowerPoint Presentation Slides (PDF)
  + Evaluation Letter from Dr. Ozment
  + Source code or link to source code with instructions for running program



[Syllabus Checker Work Plan](https://pennstateoffice365-my.sharepoint.com/:x:/g/personal/jfd5838_psu_edu/EULT1djZkwhAh09H3dYWbZoBZPg-Li8uk3EME2EB5I7Zsg?e=WnXspZ)

We continue meeting with our client, Dr. Ozment, with our next meeting scheduled for 11/10//2025. Our goal is to make sure we’re consistently meeting our client’s expectations and refining our project based on their feedback.

Test Strategy

Unit Test - Tests will verify that individual program components work correctly before integration. Major modules to be tested include:

* PDF extraction - confirm readable text is obtained from uploaded files
* Sentence parsing & cleaning - verify text is properly segmented and filtered
* Content detection - ensure required syllabus sections are correctly identified
* Readability assessment - validate scoring for Flesch and Gunning Fog
* Spell checking - confirm detection of misspellings and exemptions
* Scoring logic - verify points and penalties are calculated properly

Unit testing will occur continuously, with failures logged and corrected before integration.

Integration Test - Testing will ensure that modules operate correctly when combined. Key integrations include:

* PDF -> text -> analysis pipeline
* Model scoring -> content evaluation -> recommendations
* Backend -> GUI display + threading
* Filename parsing -> metadata reporting

Testing will take place as components are merged. Issues discovered during integration will be documented and resolved to ensure smooth system behavior.

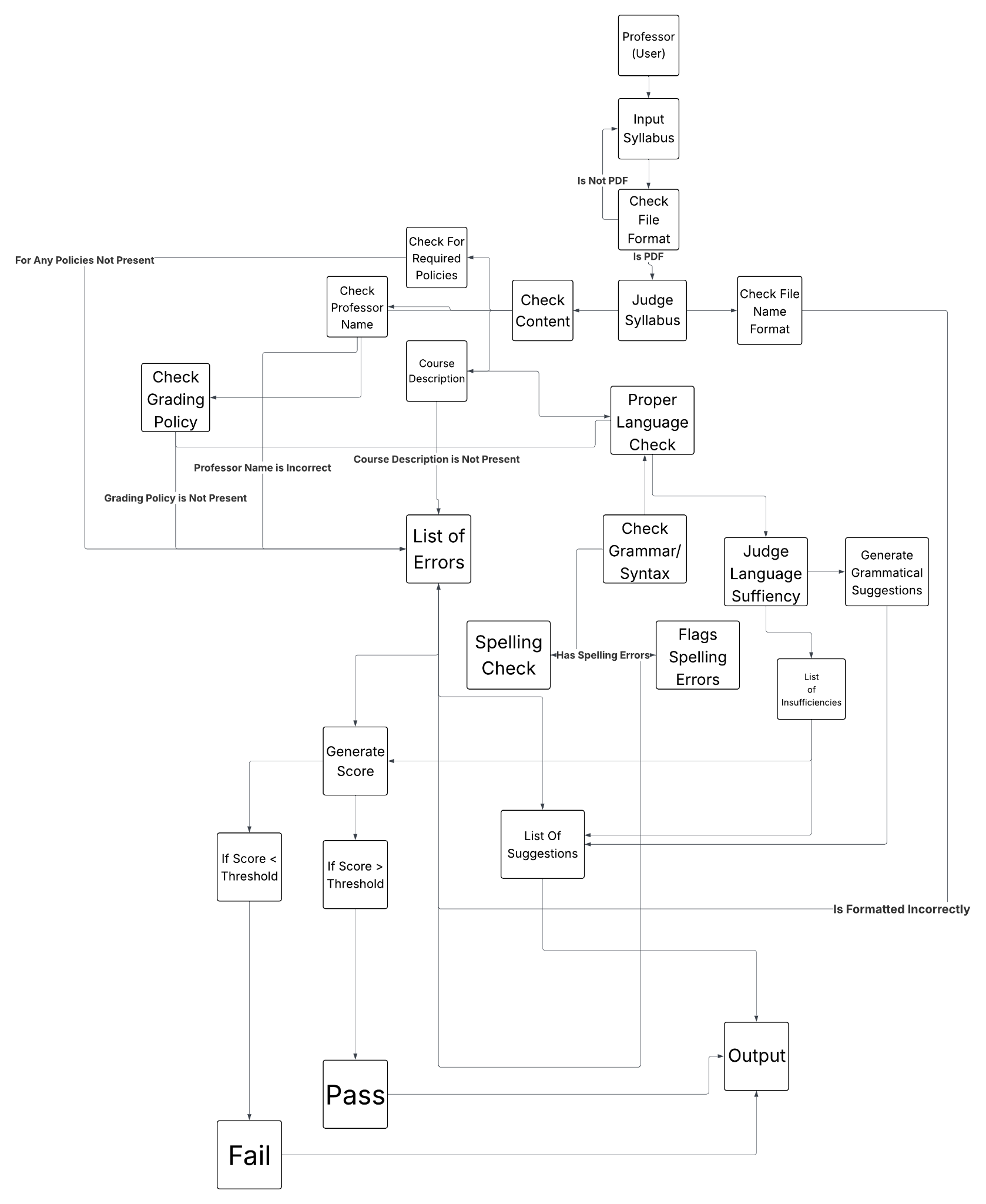
System Test - Once the software has been completed, we will run a system test to ensure everything works as expected. The main areas we will test are:

1. Functional Testing:

* Check that the program can load different PDF files
* Check that the program will not accept non PDF files
* Make sure analysis runs without crashing
* Confirm that the GUI prints the full report correctly

1. Usability / Learnability Testing:
   * Have friends/classmates try the GUI
   * Ensure that the GUI is clear to navigate and output is easy to read
   * Confirm that error messages make sense to end-user
2. Accuracy Testing:
   * Test multiple syllabi from different departments
   * Check if the program correctly detects required sections
   * Compare the output to manual checks

UML Class Diagram

****

**Figure 5. UML Class Diagram**

The program prompts the user to input a syllabus as a PDF file. It will first check to ensure that the file format is correct. If it is not, it will reject the input and ask for a new one. If it is, then the program will continue. The program then diverges into performing many analysis tasks on the input, such as required policies, name, spelling and grammar, and general readability. The program then produces a list of suggestions based on the results of these analyses, as well as generates a score to rate the input. The program will then output the input’s score, and whether they pass or fail the check. Some aspects of the input may cause automatic failure, such as missing required content. After receiving the score on the output, it will also provide the list of suggestions on how to improve the syllabus, regardless of score or pass/fail.

References

*Penn State Abington Faculty Handbook 2025 - 2026*

[*Syllabus Requirements*](https://senate.psu.edu/faculty/syllabus-requirements/)

[*43-00 Syllabi | University Faculty Senate*](https://senate.psu.edu/students/policies-and-rules-for-undergraduate-students/43-00-syllabi/)

*CMPSC 487W: Software Engineering and Design Syllabus*

[*Grammarly: Free AI Writing Assistance*](https://www.grammarly.com/?msockid=2a3c34364751602b36ff224746c561c4)

[*sentence-transformers · PyPI*](https://pypi.org/project/sentence-transformers/)

[*Pretrained Models — Sentence Transformers documentation*](https://sbert.net/docs/cross_encoder/pretrained_models.html)

[*RapidFuzz · PyPI*](https://pypi.org/project/RapidFuzz/)

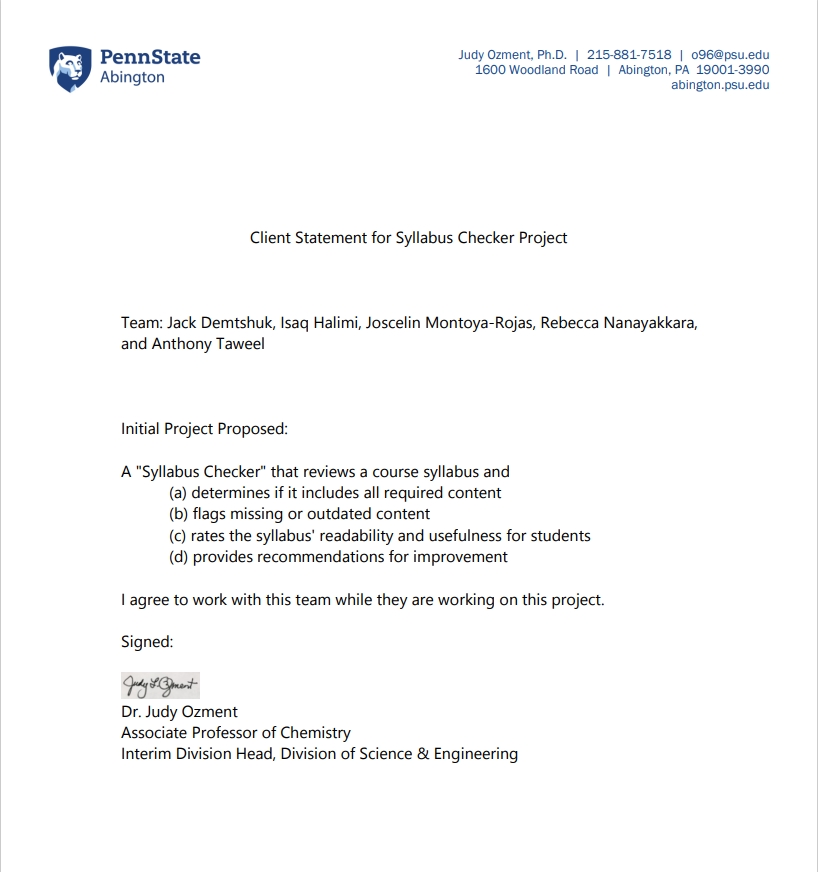
[*Flesch–Kincaid readability tests - Wikipedia*](https://en.wikipedia.org/wiki/Flesch%E2%80%93Kincaid_readability_tests)

[*Flesch Reading Ease and the Flesch Kincaid Grade Level – Readable*](https://readable.com/readability/flesch-reading-ease-flesch-kincaid-grade-level/)

[*Gunning fog index - Wikipedia*](https://en.wikipedia.org/wiki/Gunning_fog_index)

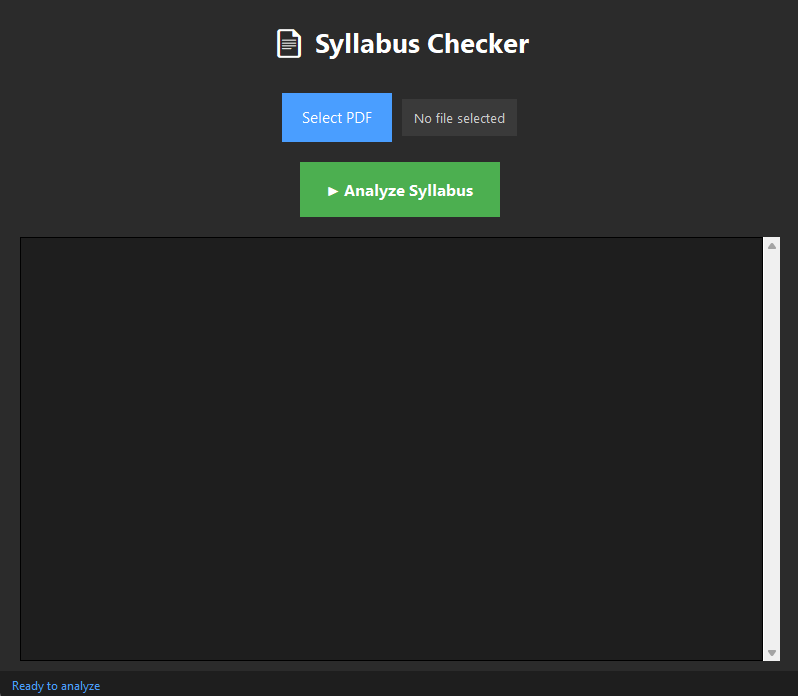
[*textstat · PyPI*](https://pypi.org/project/textstat/)

Appendix



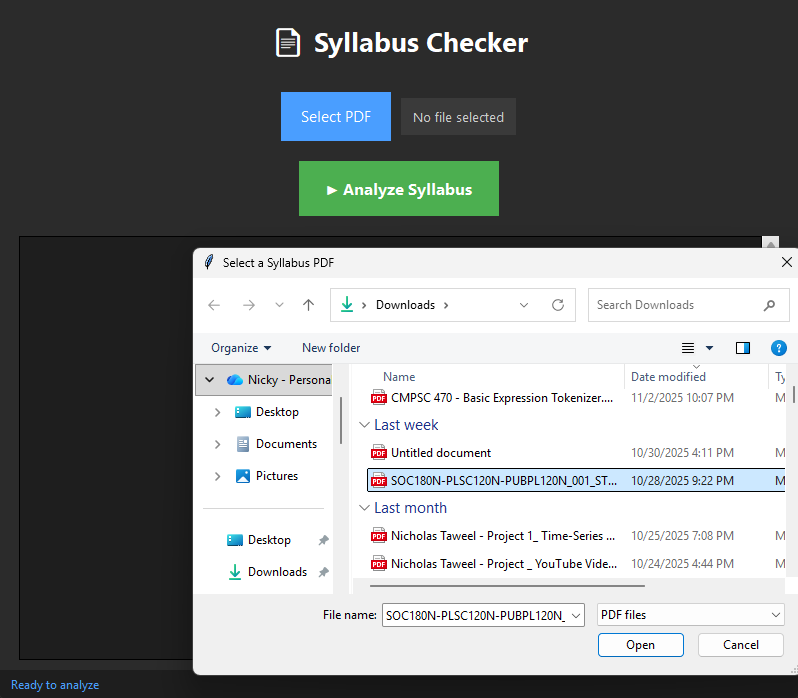
**Output Test Case**

**Default Screen**

****

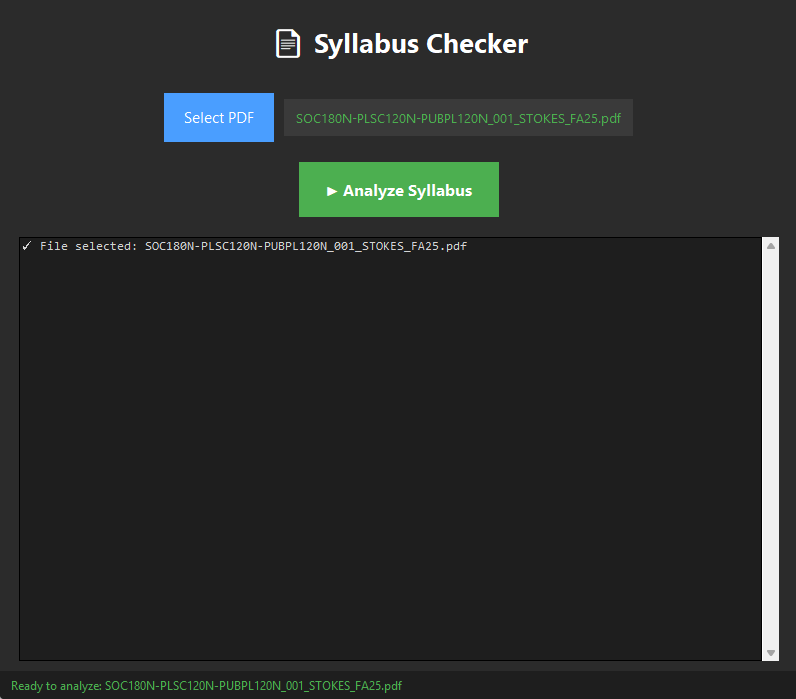
The default screen of the program once it is opened. A PDF must be selected before any analysis occurs

**Selecting File**

****

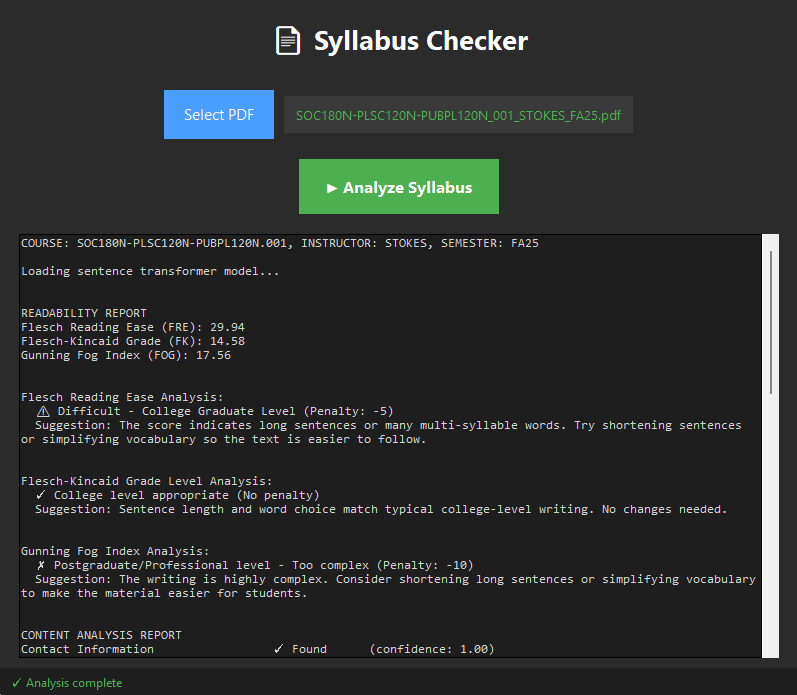
After clicking “Select PDF,” a File Explorer window is opened, to which a syllabus PDF can be selected.

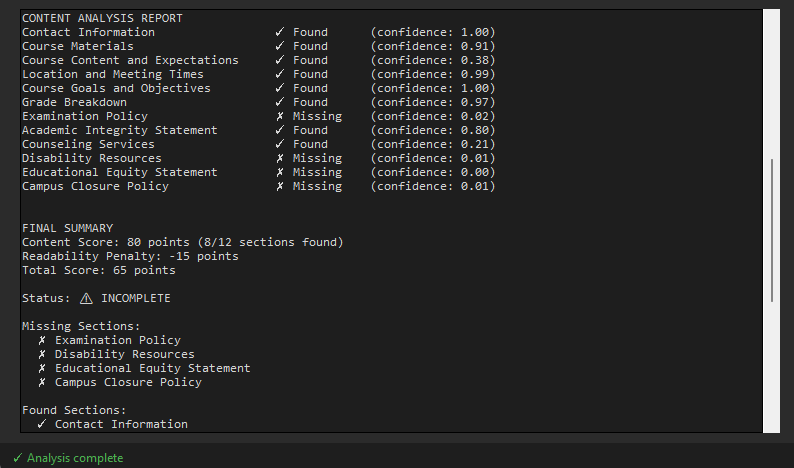
**Loading File**

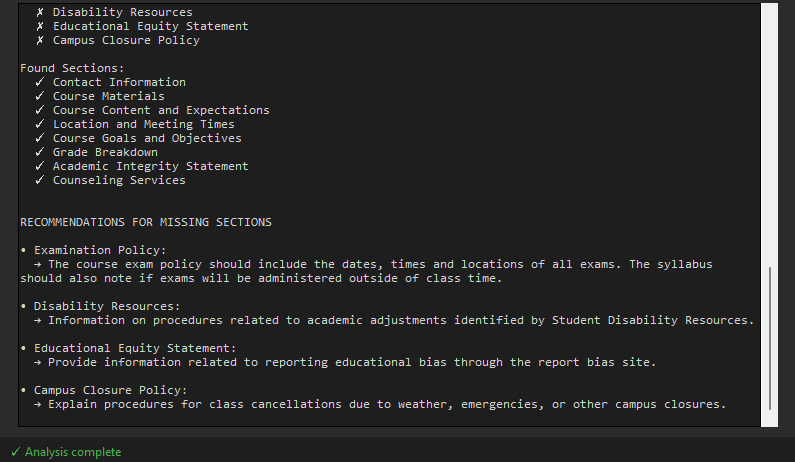
****

A syllabus will be loaded into the program after being selected. Then, the user may press “Analyze Syllabus” to begin the analysis process.

**Analysis**

****

****

****

The output windows details which sections were found and which were missing. The quality of found sections are also judged, shown by the “Readability Report.” Recommendations appear at the bottom of the window, for possible improvements to be made in the syllabus.