Submitted to,

Amit Kumar Mondal

Associate Professor

Computer Science & Engineering Discipline

Khulna University, Khulna

Submitted By,

Sourav Shome

Student ID: 2102017

Eshrat Jahan Esha

Student ID: 210233

Pushpita Chakma

Student ID: 210240

Computer Science & Engineering Discipline

Khulna University, Khulna

Generic Checklist for Code Review

Generic Checklist for Code Reviews:

Structure

	Description of Item	Pass	Fail	
1	Does the code completely and correctly implement the design?		✓	There are some buttons that does not have the functionality.
2	Does the code conform to any pertinent coding standards?	✓		Follows naming conventions.
3	Is the code well-structured, consistent in style, and consistently formatted?		✓	The code is not well structured and lack of modularity.
4	Are there any uncalled or unneeded procedures or any unreachable code?	✓		There are no indications of unreachable code.
5	Are there any leftover stubs or test routines in the code?	✓		There are no indications of leftover stubs or test routines, indicating that the code is clean and focused.
6	Can any code be replaced by calls to external reusable components or library functions?	✓		No relevant components identified.
7	Are there any blocks of repeated code that could be condensed into a single procedure?		✓	There are some repetitive patterns in the code, indicating opportunities for refactoring to condense repeated code into reusable procedures.
8	Is storage use efficient?	✓		As the code appears to handle file I/O and text manipulation efficiently without significant storage concerns.
9	Are symbolics used rather than “magic number” constants or string constants?	✓		Symbolic constants are used appropriately, enhancing code readability and maintainability.
10	Are any modules excessively complex and should be restructured or split into multiple routines?		✓	Modules “Gui.Py” contains overly complex functions, need to split.

Documentation

	Description of Item	Pass	Fail	
1	Is the code clearly and adequately documented with an easy-to-maintain commenting style?	✓		Comments are present throughout the code, providing clear explanations and enhancing readability.
2	Are all comments consistent with the code?	✓		Comments are consistent with the code, ensuring that they accurately reflect the functionality and logic implemented in the corresponding sections of code.

Variables

	Description of Item	Pass	Fail	
1	Are all variables properly defined with meaningful, consistent, and clear names?	✓		All variables are properly defined with meaningful, consistent, and clear names, ensuring clarity and readability of the code.
2	Do all assigned variables have proper type consistency or casting?	✓		All assigned variables have proper type consistency or casting, maintaining data integrity and preventing type-related errors.
3	Are there any redundant or unused variables?	✓		All assigned variables have proper type consistency or casting, maintaining data integrity and preventing type-related errors.

Structure				
	Description of Item	**Pass**	**Fail**	
1	Does the code follow the style guide for this project?			
2	Is the header information for each file and each function descriptive enough?		✓	There are no header comments in the file to evaluate.
3	Is there an appropriate number of comments? (frequency, location, and level of detail)		✓	There are not enough comments in each file, so there's no commentary on their frequency, location, or level of detail.
4	Is the code well structured? (typographically and functionally)	✓		The code structure appears to be fine.
5	Are the variable and function names descriptive and consistent in style?		✓	Variable and function names are inconsistent and not always descriptive, which can make it harder to understand their purpose.
6	Are "magic numbers" avoided? (use named constants rather than numbers)	✓		There are no magic numbers present in the provided code.
7	Is there any “dead code” (commented out code or unreachable code) that should be removed?	✓		Here is no dead code in the provided section of the file.
8	Is it possible to remove any of the assembly language code, if present?	✓		There is no assembly language code in the provided section of the file.
9	Is the code too tricky? (Did you have to think hard to understand what it does?)		✓	It's not overly complex or difficult to understand. But Gui.py file seems complex enough.
10	Did you have to ask the author what the code does? (code should be self-explanatory)		✓	Without proper documentation and clear naming conventions, it's difficult to discern the code's purpose without additional context.

Architecture

	Description of Item	Pass	Fail	
1	Is the function too long? (e.g., longer than fits on one printed page)		✓	Gui.Py file contains too long functions.
2	Can this code be reused? Should it be reusing something else?	✓		the code can be reused. Many functions are designed to perform specific tasks such as opening files, saving content, counting words, etc. These functions can be reused in other parts of the application or even in different projects.
3	Is there minimal use of global variables? Do all variables have minimum scope?	✓		The code minimizes the reliance on global variables and ensures that variables are appropriately scoped, promoting better code organization, readability, and maintainability.
4	Are classes and functions that are doing related things grouped appropriately? (cohesion)		✓	Classes and functions that are doing related things are grouped appropriately. For example, functions related to file operations (**Open, Save, SaveAs)**are grouped under the **File** menu, while functions related to editing operations(**CopyAll,Paste,Clear, Undo,Redo, Highlight, ToggleAutosave, CountWords**) are grouped under the **Edit** menu. This organization enhances code cohesion and readability. But in the **Gui.py** file presentation and business logic collapse with each other that need to be organized.
5	Is the code portable? (especially variable sizes, e.g., “int32” instead of “long”)	✓		The code appears to be portable, as it primarily uses the Tkinter library for GUI development, which is cross-platform and compatible with different operating systems.
6	Are specific types used when possible? (e.g., “unsigned” and typedef, not just "int")	✓		Specific types are used when possible. For example, **str** is used to represent file paths (**self.file_path**), **bool** for boolean variables (**self.autoSave, self.isSaved, self.isEdited**), and **int** for counting words.

7	Are there any if/else structures nested more than two deep? (consecutive “else if” is OK)	✓		There are no if/else structures nested more than two deep in the provided code snippet. Control flow seems to be straightforward and easy to follow.
8	Are there nested switch or case statements? (they should never be nested)	✓		Here are no nested switch or case statements in the provided code snippet.

Variables

	Description of Item	Pass	Fail	
1	Does the code avoid comparing floating-point numbers for equality?	N/A	N/A	Since the code doesn't involve numerical computations or comparisons of floating-point numbers, there's no explicit need to compare floating-point numbers for equality.
2	Does the code systematically prevent rounding errors?	N/A	N/A	Given the absence of explicit numerical computations or operations prone to rounding errors, there's no direct need to systematically prevent rounding errors in the code.
3	Does the code avoid additions and subtractions on numbers with greatly different magnitudes?	N/A	N/A	Since the code doesn't involve numerical computations or operations on numbers, there's no scenario where additions or subtractions with greatly different magnitudes would occur.
4	Are divisors tested for zero or noise?	N/A	N/A	The code doesn't perform division operations involving divisors that could potentially be zero or noisy.

Loops and Branches

	Description of Item	Pass	Fail	
1	Are all loops, branches, and logic constructs complete, correct, and properly nested?	✓		The code appears to have correct and properly nested logic constructs, including loops and branches. The structure follows standard GUI event-driven programming patterns.
2	Are the most common cases tested first in IF- -ELSEIF chains?	N/A	N/A	Since the code doesn't contain extensive IF-ELSEIF chains, this item is not applicable.
3	Are all cases covered in an IF- -ELSEIF or CASE block, including ELSE or DEFAULT clauses?	N/A	N/A	Since the code doesn't include IF-ELSEIF or CASE blocks with multiple cases, this item is not applicable.
4	Does every case statement have a default?	N/A	N/A	As there are no case statements in the provided code, this item is not applicable.
5	Are loop termination conditions obvious and invariably achievable?	✓		Loop termination conditions, if any, are apparent and achievable based on the logic implemented in the code.
6	Are indexes or subscripts properly initialized, just prior to the loop?	N/A	N/A	Since there are no explicit loops with index variables, this item is not applicable.
7	Can any statements that are enclosed within loops be placed outside the loops?	N/A	N/A	Since there are no explicit loops in the provided code, this item is not applicable.
8	Does the code in the loop avoid manipulating the index variable or using it upon exit from the loop?	N/A	N/A	Since there are no explicit loops with index variables, this item is not applicable.

Maintainability				
	Description of Item	**Pass**	**Fail**	
1	Does the code make sense?	✓		Overall, the code makes sense and appears to function as intended for a text editor GUI application.
2	Does the code comply with the accepted Coding Conventions?	✓		The code generally follows Python coding conventions, including the use of consistent naming conventions and indentation.
3	Does the code comply with the accepted Best Practices?	✓		While it's subjective to determine the "best" practices, the code follows common practices for developing GUI applications using the **Tkinter** library in Python.
4	Does the code comply with the accepted Comment Conventions?	✓		The code includes comments at the start of each file and function, providing explanations of their purpose and usage. Additionally, comments are used throughout the code to explain complex logic or functionality.
5	Is the commenting clear and adequate?	✓		Comments are clear and adequate, providing insights into the code's functionality and aiding readability. Each file has introductory comments, and functions are documented appropriately.
6	Are ideas presented clearly in the code?	✓		Yes, the ideas are presented clearly, and the code is organized logically, making it easy to follow the flow of execution.
7	Is encapsulation done properly?	✓		Encapsulation appears to be appropriately implemented, with functions and classes encapsulating related functionality.
8	Is the code not too complex?	✓		The code complexity seems reasonable for a GUI application, with functions and classes structured to handle specific tasks.

9	Are there no unnecessary global variables?		✓	There are a few global variables used in the gui.py file (**file_path, content, isSaved, isEdited, autoSave**). While some of these may be necessary for the application's functionality, minimizing the use of global variables is generally considered good practice.
10	Is the reading order in source code from top to bottom?	✓		The code is organized in a top-to-bottom manner, with functions, classes, and imports appearing in a logical order.
11	Are there unused variables or functions?	✓		There don't appear to be any unused variables or functions in the provided code.

Reusability

	Description of Item	Pass	Fail	
1	Are all available libraries being used effectively?	✓		The code effectively utilizes the **Tkinter** library for creating the GUI components and handling user interactions.
2	Are available **OpenMRS** util methods known and used?	N/A	N/A	The code does not appear to utilize any **OpenMRS**-specific utility methods, so this criterion is not applicable.
3	Is the code as generalized/abstracted as it could be?		✓	The code is tailored to the specific requirements of a text editor GUI application and could be further abstracted or generalized to increase its reusability in other contexts.
4	Is the code a candidate for reusability?		✓	It serves its purpose well as a text editor, but its reusability in other contexts may be limited.

Robustness				
	Description of Item	**Pass**	**Fail**	
1	Are all parameters checked?	✓		The code seems to adequately check parameters where necessary to ensure validity and prevent errors.
2	Are error conditions caught?		✓	Error conditions are not appropriately handled throughout the code.
3	Is there a default case in all switch statements?	✓		All switch statements appear to include a default case, ensuring that all possible cases are covered.
4	Is there non-reentrant code in dangerous places?	✓		It serves its purpose well as a text editor, but its reusability in other contexts may be limited.
5	Is the usage of macros proper? (Readability, complexity, portability...)	✓		There are no macros visible in the provided code section. If macros were used, their usage should be evaluated based on readability, complexity, and portability.
6	Is there unnecessary optimization that may hinder maintainability?	✓		The code does not appear to contain unnecessary optimizations that could hinder maintainability. However, this would require a more detailed analysis of the codebase to confirm.

Error Handling

	Description of Item	Pass	Fail	
1	Does the code comply with the accepted Exception Handling Conventions?		✓	The code does not include exception handling, which is essential for robust error management and fault tolerance.
2	Does the code make use of exception handling?		✓	Exception handling is not implemented in the codebase, which could lead to runtime errors and unexpected behavior.
3	Does the code simply catch exceptions and log them?	N/A	N/A	
4	Does the code catch general exception?	N/A	N/A	
5	Does the code correctly impose conditions for "expected" values?	N/A	N/A	
6	Are input parameters checked for proper values (sanity checking)?	N/A	N/A	
7	Are error return codes/exception generated and passed back to the calling function?	N/A	N/A	
8	Are error return codes/exceptions handled by the calling function?	N/A	N/A	
9	Are null pointers and negative numbers handled properly?	N/A	N/A	
10	Do switch statements have a default clause used for error detection?	N/A	N/A	

11	Are arrays checked for out-of-range indexing? Are pointers similarly checked?	N/A	N/A	
12	Is garbage collection being done properly, especially for errors/exceptions?	N/A	N/A	
13	Is there a chance of mathematical overflow/underflow?	N/A	N/A	
14	Are error conditions checked and logged? Are the error messages/codes meaningful?	N/A	N/A	
15	Would an error handling structure such as try/catch be useful? (depends upon language)	N/A	N/A	