Final Report

Design of a Simpler Code Editor That Improves Focus and Learning

Malak Sefrioui, 40252907
William Charron-Boyle, 40264407
Tanzir Hoque, 40210275
Zi Lun Li, 40191860

SOEN 357
Due November 29th 2024

Abstract
In this project, we believed that newcomers to programming feel overwhelmed by modern day integrated development environments, which hampers their learning experience. We aimed to offer a minimalistic and visual code editor that offers core functionalities while using visuals to link theory concepts to coding. The project was realized by utilizing GitHub as a hosting service for the code for our app, and developed with C, more specifically, the Windows API, with modern day computers using Windows 11. Results of our surveys showed an overwhelming interest in Edit#, as well as its core features.

Introduction
For our project, we aim to simplify code and text editors to make them more accessible and less overwhelming for users, especially those who are new or relatively unfamiliar with programming. Traditional code editors such as IDE like Eclipse, while they offer extensive complex tools and functionalities, often present a steep learning curve with an abundance of features, configurations, and complex user interfaces that can intimidate beginners or non-technical users. We want to address this problem by reinventing code and text editors, going back to the good old days of notepad so that it can reach a broader audience, including beginner programmers, teachers, students, and individuals who may be less technical when it comes to programming. Our mission is to streamline the user experience process when coding by focusing on the simplicity, clarity, and essential functionalities of the text editor. By solving this problem, we aim to make coding and text editing more approachable, empowering users to learn, create, and innovate without feeling discouraged by overly complex tools. This will foster an environment of inclusivity in technology, reducing barriers to entry and accelerating learning for a diverse audience.

Research Question
Our research question consists of understanding the pain points of an audience new to programming when exposed to modern code editors. What exactly makes complex code editors such as Visual Studio Code less approachable for non-technical users? Why is this an issue and how can we solve this so that more people develop an interest in coding rather than feeling rejected to learn it because of the sheer amount of information modern code editors throw at you first hand? To address these questions, we develop a solution to solve the issue mentioned previously using UI/UX/IxD design concepts learned from Concordia University's SOEN 357 class concepts by simplifying the code editor to its essential features. Thus, removing features such as dependency packages, extensions, etc. that are normally found in modern tools and simply focus on writing code, will inherently facilitate ease of use and reduce the cognitive load that can often be overwhelming for a first time experienced user.

Hypothesis
We believe that users who are new to programming may feel less overwhelmed if we develop a code editor with a minimalist interface. By condensing the text editor to its core features, this approach enables the users to increase productivity, interest and have a better understanding of programming rather than allocating time for setting up environments or fixing issues with library imports. By doing so, we would increase new programmers to think more rationally thus

improving their skill set by putting their focus on a singular aspect of programming which can directly increase learning curve and potential interest to more advanced coding as a result. We want to be able to keep the barrier of entry low for our target audience to make coding more approachable for non-technical users. As they have taken the first step to get exposed to programming, they will naturally lean towards more advanced code editors as their learning curve and interest continues to grow steadily in a linear progression. We believe this hypothesis will hold true because reducing unnecessary complexity, providing clear instructions, and emphasizing ease of use can lower cognitive load, making the tool more accessible. In order to validate our hypothesis, we will measure outcomes such as the completion time of tasks, the user's satisfaction, the ease of use, the learning curve and the interest developed while using the text editor. We expect that our design for simplicity will greatly alleviate the pressure and daunting task of programming for beginners and increase their efficiency of learning the basics overall.

Project goal

Our project goal is to create a minimalistic code editor that offers the only core functionalities while removing the extra features that are found in modern code editors in order to reduce the cognitive load of non-technical users and enhance their first time experience learning of programming. By doing so, we hope to increase the interest and the satisfaction of coding to a broader audience.

Related Works

The simplicity that Edit# promotes aims to fix the issues of other code editors. These issues are present in some integrated development environments (IDEs) as well. An example of such software is Microsoft Visual Studio 2022. This tool incorporates a plethora of functionalities, including some that are as follows:

- A text editor with a configurable appearance and syntax highlighting,
- IntelliSense, an autocompletion mechanism that guesses the user's intent in function of context. For instance, this system takes local symbols into consideration when displaying possibilities for input,
- Live compilation warnings and errors for ill-formed code as the user codes,
- A built-in compiler minimally compliant with C89 and C++11,
- The integration of GitHub to update a project according to the latest repository changes,
- A set of refactoring functionalities for maintaining codebases.

The versatility of this tool does not necessarily endow users with maximum efficiency however. This diversity of options can overwhelm users wishing to learn how to code. Furthermore, a high-level interface for writing and compiling code can obscure important details. Beginners failing to understand these details may struggle to assimilate important concepts. An example of such a detail is syntactic particularities. Suggesting corrections to fix, say, variable type errors provides poor feedback to learners. Excessive reliance on this mechanism can prevent learners from understanding what said corrector detects. Novice coders ought to make errors in their journey of learning how to code. Otherwise, learners might not know how to diagnose issues when no corrector is available. Microsoft Visual Studio and Edit# are incomparable due to their significant difference in scale. However, learning environments relying on either tool would be

more comparable between each other. The design of Edit# intends to separate the compiling and coding processes for learners. This separation naturally allows novices to proceed only once they understand their errors. One edit to a source file that causes a compilation error provides valuable feedback. The coder can then understand why their procedure is incorrect and adjust accordingly. This learning process can only work if the compiler provides clear feedback regarding errors. Edit# does not provide any hints regarding the presence of ill-formed code. It must be clear at this point that Edit# does not bear a compiler. The coders can choose one, like GCC or Clang, according to their needs. Edit# only offers an interface to edit the contents of source files, like Notepad++. The latter shares some of the aspects that characterizes the former. Notepad++ is a simple code editor with features that are as follows:

- A display with configurable colors, font, and syntax highlighting,
- An autocompletion mechanism suggesting completions in function of the contents of a file. The user can configure this mechanism to only suggest reserved keywords for programming languages. Notepad++ supports over fifty different programming language syntaxes,
- A “find and replace” function for replacing all instances of a string with another.

This editor does fall short for one aspect, which is GitHub integration. The lack thereof adds some overhead to update files in function of recent changes. Edit# covers this missing functionality to maximize efficiency in projects using GitHub’s repository system. Furthermore, Notepad++ only supports the Windows operating system. This restriction does allow the editor to take advantage of this operating system’s particularities. This specialization gives the performant and light-weight character of this code editor. Edit# could take this concept a step further to offer system-specific versions of itself. That is, variants of the editor can exist for Windows, MacOS, and Linux. Such support would require careful monitoring of the behaviors of all three variants. All three versions would require to behave similarly to attain consistency. This approach can be demanding, but allows each implementation to exploit its host OS. Thus, Edit# would perform adequately even on the poorest-performing computers regardless of OS. Some frameworks exist to support applications that aim for this goal, like Qt. This library provides cross-platform windowing and rendering functionalities for applications. Overall, this flexibility would help Edit# prevail among non-IDE code editors.

Methods

The development of Edit# requires a version control service so that many people may be able to contribute and work on the code editor. As this project was developed using C++ with the Qt library, the operating system of choice among all contributors is Windows 11. Furthermore, the developers will need to meet the minimum system requirement in order to use the application. The minimum system requirement for development calls for having 8GB of random-access memory (RAM), preferably a solid state drive with at least 500GB of storage, a dual-core central processing unit (CPU) with at least 2 GHz of maximum speed, a motherboard that can accept an Intel or AMD CPU, and finally a power supply that can provide at least 500W of power [1]. CodeLite is an appropriate integrated development environment to use to develop Edit#, as it both supports C++ and Git, although the latter is through a plugin [2]. One reason why C++ was chosen was because it offers the ability to have Edit# excel in performance while maintaining its lightweightedness. In fact, there is research that shows that C++ prevails in performance and

excels at compiling [3] [4]. Furthermore, C++ is a cross-platform programming language. Although the compilation process may differ from one operating system to another, ultimately it remains cross-platform, allowing Edit# to reach the largest target audience. This problem is solved by implementing frameworks such as Qt. It is a library that handles windowing and rendering for Windows 11, MacOS, and Linux. Before implementing the application, we can use tools like Figma in order to create wireframes and prototypes of Edit# to better proceed during development. Although a code editor would be lacking in substantial colors and styles, it is still important to have a visual example of the editor, as well as for any menus that were implemented in the editor, as those require deep and extensive design decisions. This project opted to develop a prototype of this sort using the Windows API. The programming language for this first iteration of the editor was C. This prototype rests in a Git repository [5] available via GitHub.

Evaluation

To gather the data we need to answer our research question, we will be conducting 2 studies: a feasibility study, at the beginning of the project, and a later stage study at the end of the project. The first questionnaire is going to act as a feasibility study. Not only will the data collected from the feasibility study allow us to know if the project is feasible as the name suggests, but it will show us if the project will garner a positive reception. The subject/questions we want to address in the survey would be to, first, make sure the goal of our project is wanted by the targeted audience. Second, after the initial research on other products, we would have found which features are overwhelming and confusing and which are very helpful to beginners. These assumptions we would have deducted can then be checked to see if the target audience agrees with them or not. Third, we can also ask questions more related to the user themselves to have a better understanding of them. Finally, the last survey can be done after the product has been fully created. This survey will show us if our hypothesis at the beginning of the project was right or wrong. This survey will measure the 5 outcomes that will validate or not our hypothesis: the time-to-task completion, the user satisfaction, the ease to use as well as the learning curve and interest accumulated throughout the user's use of the final product. The time-to-task completion can be measured with a checkbox question. The user will be asked how long it took them to complete a certain task such as opening a file. Then, they will have a multiple-choice question including the following : "less than 1 second", "~10 seconds", "~30 seconds", and we would have an "More than a minute" option for answer that might be different from the given options such as "the feature did not work". The user satisfaction can be measured with a semantic scale going from 1 to 5 with 1 being the worst experience they had with any code editor and with 5 being the best experience they had with any code editor. The ease to use/learning curve can be measured with a semantic scale as well going from 1 to 5. Finally, the interest can also be measured from 1 to 5 on a likert scale.

We will know if we succeeded at this project through the measures of outcomes collected during the later stage study. The time-to-task question will have to be answered by at least half of the respondents with the option "between 10 and 15 seconds". The user satisfaction has to be answered by at least half of the population with 3 and up on the scale. The ease as well as the learning curve and interest all have to be 4 and up by more than half of the population.

To get the highest response rates to all the questionnaires, those will be done online through Google forms. We will use the snowball method as well to reach that goal. We will be focusing on the answers of the respondents fitting our idea of the target audience, but we will welcome any individual outside of that audience to get a bigger picture of our subject.

What we did - Feasibility

We created a 12-question survey (see Appendix A) comprising two sections: the first section includes questions related to the product/project idea while the second has questions concerning the user. The first section had one question about the user supporting the project idea and the rest were statements about features the user agreed or disagreed with. Those were mandatory. However, the second section was optional if the questions were too sensitive to answer. The questions were to understand the motivation, the educational background, age and more.

What we did - Later-stage study

For the prototype evaluation, we included four questions (see Appendix B) measuring the following: the user satisfaction, ease of use/learning curve, interest over the product, and time-to-task completion. The first three questions are evaluated on a likert scale from 1 to 5 while the last is a multiple-choices question.

Results

Surveys Responses - Feasibility study

The responses of the feasibility study were overwhelmingly positive throughout the diverse demographic (see Appendix C). The majority of the responses (80%) are interested in our project idea. A majority of respondents seemed to agree that the following features were attractive/useful and not overwhelming : syntax highlighting, identifying the lines of code, and customizing the application. The majority of respondents (85%) also agreed that a github feature would only complicate the code editor and make it overwhelming. For the demographic of the respondents, we were able to collect a diversified population in age, skill level, and motivation for coding. This would mean that the statements we made for the features' evaluation are a common opinion across age and education/skill level.

Surveys Responses - Later-stage study

Again, the response was positive towards the later-stage study (see Appendix D). Even though the prototype lacked some features, the user satisfaction was still positive. 60% of the population rated their user satisfaction 4 on a scale of 5, and 40% rated it 3 out 5. 80% rated their ease of use and ability to learn a 5/5 while the rest rated it a 4/5. The respondents' interest in the product was also positive with 60% claiming their interest to be a 5 on a scale of 5. The rest claimed it to be a 4/5. The performance result collected out of the time-to-task completion question came back to be also very positive since 100% of the respondents said the task took them less than a second to perform.

By comparing these results to the standards we gave ourselves in the evaluation section, we can assume our project to be an overall success. The threshold we had set to measure our success was the following for each question :

- User satisfaction : more than 50% answered at least 3/5
- Ease of use : more than 50% answered at least 4/5
- Interest : more than 50% answered at least 4/5
- Time-to-task completion : more than 50% answered “less than a second”

The results from the survey were better than where we set the threshold for our success to be. Therefore, we have succeeded in proving our hypothesis concerning the difficulty of using code editors that have too many features.

User Persona

Our first user persona is Simon Zhang (see Appendix E). He is a computer science undergraduate at Concordia University and teaches introductions into programming to high school students as a side hustle to help pay his student loans. He is able to do so because of his sheer knowledge from being exposed to programming at a young age from doing LeetCode.

Our second persona is Natasha Martinez (see Appendix F) who used to work as a bank clerk and have since retired. She now enjoys her retirement life by exploring new hobbies and happens to stumble upon programming.

These two personas were created to put into perspective the user’s point of view so that we can understand the different pain points that can occur when a user tries to use similar products to our project. This would allow us to understand the user thought process and solve those problems so that our developed text editor is able to accommodate the needs of our target audience.

Figure 1: Identification of potential frustrations among coders regarding code editors.

Here is a user journey under the perspective of our first user persona Simon Zhang, where he is attempting to figure out a solution to simplify his teaching by utilizing simpler tools such as text editor so that his students who lack general knowledge can follow along with ease. The scenario describes how he tried to utilize similar tools to fit his needs but failed.

Design

Preliminary sketches are useful for experimenting with layouts for interfaces of applications in development. They help identify the features that contribute most to the character of the interface. Later iterations of the interfaces can improve upon these features to generate optimal layouts. Figure 1 shows some initial sketches of the layout of the interfaces of Edit#.

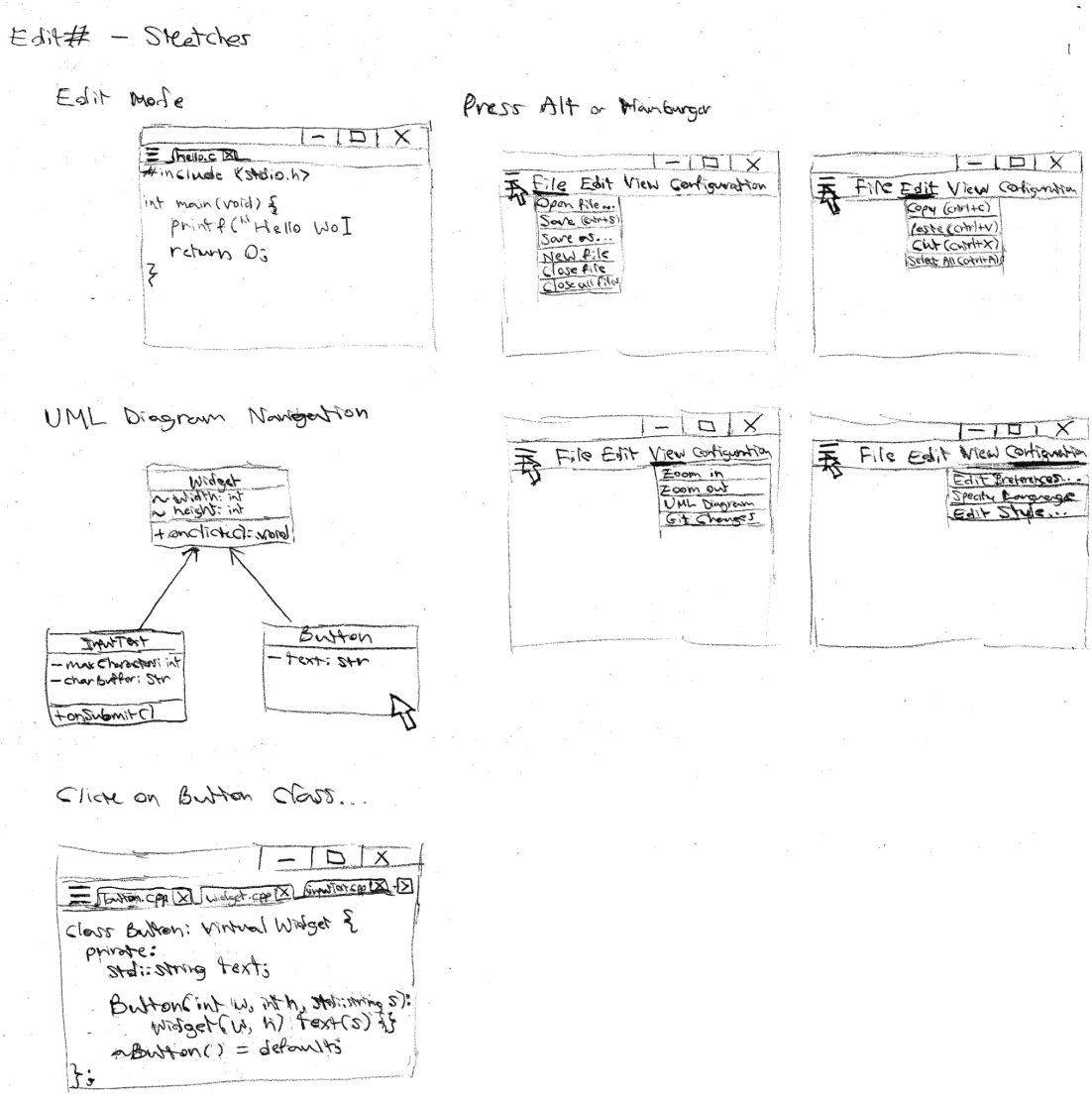

Figure 2: Preliminary determination of the layout of Edit#.

The Figma prototyping tool can help to flesh out these first layouts. This tool supports the design of wireframes for later designs to rely on. The prototyping phase used this tool to create its own set of these wireframes [6]. A user flow chart serves to describe the behavior of the final user interface. A Miro project [7] illustrates the abstraction of the interactions between Edit# and its users. This user flow chart inspires the transitions of the visual prototype for Edit#. This product aims to illustrate the final design of Edit#. This prototype takes the form of a presentation using Figma [8]. This visual prototype shows how Edit# could implement syntax highlighting for the following languages

- C,
- C++,
- Python 3.

The prototype partially demonstrates the behavior of Edit#'s Git manager and the UML navigator.

Discussion

The core features that we aimed to implement, such as numerating lines of code, syntax highlighting, a streamlined set of features for GitHub, as well as file manipulation (creating, editing, saving, deleting), were all features that almost unanimously were desired in a code editor. Furthermore, an overwhelming majority of users according to our surveys were on board for what Edit# had to offer. As such, it can be concluded from the results that our project is relevant and has a very real audience that it can offer utility towards. Furthermore, the idea of a simplified code editor is something that has very much been talked about in the past [9] [10]. The results acquired are in line with other case studies and theses that have been conducted. As such, Edit# differs from other projects within the same space by being as minimal as possible to provide a seamless user experience that lessens the learning curve by not immediately jumping to the IDE format. That being said, the scope of our project, as well as the data that has been gathered, regardless of its backed up validity, is ultimately restricted. First and foremost, as the project is in the scope of a course, one that members of this project are taking with several others. This means that we are restricted in time and scope. As such, we are not capable of implementing every feature that was discussed between members, such as code auto-completion. The implementation of such a feature would require a robust and optimized probability distribution algorithm that has statistics on what programmers are most likely to write based on the beginning of a string. This is far too large of a task with limited time, and as such, was ultimately not implemented. Furthermore, our survey data cannot reach a very large audience as this project is one of many of the same course. The range and networking potential of this project is not as realized as some research papers [9] are. Therefore, survey results show that people have been overwhelmingly receptive to the core idea of what Edit#, as well as its features, however we were not able to implement the full scope of what Edit# could potentially be due to time constraints.

Conclusion

With our initial hypothesis, we consider that a simplified and intuitive code editor can greatly improve the user experience of those who are interested in programming but that often lack the knowledge or feel overwhelmed to do so. Through our research and data collection, we realized that a big majority of the responses were interested in the idea and agreed with the features we believed to be overwhelming/easy to understand. We, therefore, proceeded with the design of the product through user personas, a user flow diagram, sketches, and used GitHub to iteratively develop the code editor. As such, our hypothesis was proved and an audience exists for Edit#. While we feel content with what was ultimately achieved, what could be done next would be to work on code auto-completion, and after that, a compiler.

References

[1] NewEgg. *Power Supply Calculator*, 2024, https://www.newegg.com/tools/power-supply-calculator.

[2] CodeLite. “Git Plugin”. *CodeLite Documentation*, 2024. https://docs.codelite.org/plugins/git/.

[3] Fourment, Mathieu, and Michael R Gillings. “A Comparison of Common Programming Languages Used in Bioinformatics.” *BMC Bioinformatics* 9, no. 1 (December 2008): 82. https://doi.org/10.1186/1471-2105-9-82.

[4] Aruoba, S. Borağan, and Jesús Fernández-Villaverde. “A Comparison of Programming Languages in Economics.” Cambridge, MA: National Bureau of Economic Research, June 2014. https://doi.org/10.3386/w20263.

[5] GitHub. “Edit-sharp.” 2024. https://github.com/jws412/edit-sharp.

[6] Figma. “SOEN 357 Final Project Wireframes.” 2024. https://www.figma.com/design/UAryX2ugMLhrsk5K2KcsXq/SOEN-357-Final-Project-Wireframes?t=rvCa1HOGetj1tSs5-1.

[7] Miro. “SOEN 357 Final User Project Flow.” 2024. https://miro.com/app/board/uXjVPofHWhg=/?share_link_id=424547655833.

[8] Figma. “SOEN 357 Final Project Visual Prototype.” 2024. https://www.figma.com/proto/lEj693eilpi8cgjCDGCJAO/SOEN-357-Final-Project-Visual-Prototype?node-id=0-1&t=HVRB5Gj8xYLthhHg-1.

[9] Y. Bosse and M. A. Gerosa, “Why is programming so difficult to learn? Patterns of Difficulties Related to Programming Learning Mid-Stage,” *SIGSOFT Softw. Eng. Notes*, vol. 41, no. 6, pp. 1–6, Jan. 2017, doi: 10.1145/3011286.3011301.

[10] Mateas, Mario-Mihai, and Cosmin Marsavina. “Pie: A General-Purpose Code Editor Focused on Simplicity.” In *2024 IEEE 18th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, 000291–96. Timisoara, Romania: IEEE, 2024. https://doi.org/10.1109/SACI60582.2024.10619920.

Appendices

Appendix A: Google Forms. “Feasibility study - Edit#.” 2024. https://forms.gle/xnKDozmDE38Uafey6.

Appendix B: Google Forms. “Later-stage study - Edit#.” 2024. https://forms.gle/hivbn1ZtrMdJ7taY7.

Appendix C : Google Drive, “Feasibility study - Edit#.” 2024. https://drive.google.com/file/d/1kpaixNxRlO0VX25dkgpFjk_8DBcLqy9T/view?usp=sharing

Appendix D : Google Drive, “Later-stage study - Edit#.” 2024. https://drive.google.com/file/d/1N8eubwxSXq1Za4R2IYT4WhPmZVqwLJfe/view?usp=drive_link

Appendix E : Xtensio. “Simon Zhang.” 2024. https://workspace16822758.xtensio.com/5jmjhi0i.

Appendix F : Xtensio. “Natasha Martinez.” 2024. https://workspace16822758.xtensio.com/c37tf7tq.