Algo-Mate

A. Bochman, L. Dias, N. Lamberson, & D. Lamoreaux

University of Massachusetts Lowell — Software Engineering II Prof. James Daly

Motivation

Whenever programmers need to find code snippets, design patterns, or need help understanding how a design pattern is implemented, they generally find themselves on websites like Stack Overflow[1] and CPP Reference[2] to find solutions to their problems and outlines for commonly used patterns to insert into their code. Whether a programmer is just starting to experiment with these extremely important programming standards, or has to learn them a new language, many spend a large amount of their time that could be used productively searching the internet for the proper syntax for the language they are coding in.

One issue that we face every day as programmers is the constant threat of distraction. According to Gloria Mark of the Department of Informatics at the University of California[3], it takes the average person 23 minutes and 45 seconds to return to a task fully after a distraction, and there are no stronger distractions than the vast world of the internet. A single moment of weakness, or a simple mis-click might land someone at the front page of Reddit, for instance, and they may find themselves reading click-bait articles for hours.

There is a strong need for a solution within the IDE for references on how to complete these specific objectives, and while there are currently some code completion capabilities built into Visual Studio Code, they only cover basic snippets such as loops. Algo-Mate's goal is to add more functionality by including code insertion for several common design patterns with assisted completion and helpful comments, helping users implement powerful design patterns into their code while ensuring to keep users on the right track without the risk of distraction.

Objectives

The Algo-Mate team has streamlined the insertion of common blocks of code to prevent programmers from losing focus of Visual Studio Code (VS Code) while increasing productivity. In the process, we are simplifying the implementation of design patterns, making the use of them more accessible for programmers. Whether you have used them before or are trying to implement them for the first time, Algo-Mate will streamline the process, giving comments to help a programmer step through the implementation process. In other words, we've made tedious design patterns easier to implement across multiple programming languages. Ideally for users of VS Code, gone will be the days where you transition from C++ to Java to Python back to C++

[1] https://stackoverflow.com/
[2] https://en.cppreference.com/w/
[3] https://www.ics.uci.edu/~gmark/chi08-mark.pdf

and forget the exact syntax of writing a design pattern. With Algo-Mate, the buttons, controls, and auto-completed code snippets we've implemented will increase the efficiency, productivity, and even enjoyability of writing code in the supported languages.

A simple help menu placed on the status bar at the bottom of each window will provide the user with a list of the design patterns which are implemented in Algo-Mate in each available programming language. When a user selects a language and a design pattern, they will be linked to a reputable website which provides thorough information on the use and implementation of the design pattern. This reduces the friction in trying to find the appropriate resources, allowing for the programmer to find accurate and helpful documentation when learning about the design pattern they are implementing.

Algo-Mate supports design patterns implemented in C++, Java, JavaScript, and Python. The language of the file in the editor is detected by the extension. Typing short codes or common phrases associated with a design pattern will allow the user to autocomplete a design pattern by hitting enter or clicking the desired option in the dropdown menu. Additionally, inline comments accompany the auto-completed code to provide useful information on how the design pattern works. There are also comments on what data the programmer is required to populate in order to complete the remainder of the design pattern to fit their use case.

Finally, a web page has been developed to act as a hub for the project and contains relevant information on all aspects of the project. This list includes: an **overview** with a synopsis and demo video; **members** section with information and links to each members' LinkedIn, GitHub, personal website, and email; an **updates** section for release notes, documents, and resources; and finally, **external links** are provided to private sections and the project's source code.

Approach

Algo-Mate has implemented 4 total design patterns, which are made to reach the most common and useful patterns. These design patterns are implemented in C++, Java, JavaScript, and Python for a good coverage of commonly used programming languages. The four design patterns fall into three categories, creational, structural, and behavioral. Creational design patterns focus on creating objects and initializing classes. Structural design patterns are focused on forming larger structures through the combination of classes and objects. Lastly, behavioral design patterns focus on identifying communication between objects, determining their responsibilities. Within these groups of design patterns, we have implemented:

1. Singleton (Creational)
2. Builder (Creational)
3. Adapter (Structural)
4. Observer (Behavioral)

These design patterns autofill similar to how IntelliSense works in VS Code, retrieving the templates from a JSON file that stores the keywords and their corresponding design patterns.

IntelliSense autofill works by analyzing what the user is typing and displays suggestions based on the input. For Algo-Mate, this behaves the same way, where words like “singleton” or “builder” can be typed into a file in VS Code and Algo-Mate will give the suggestion to autofill. This can be seen in Figure 1 below:

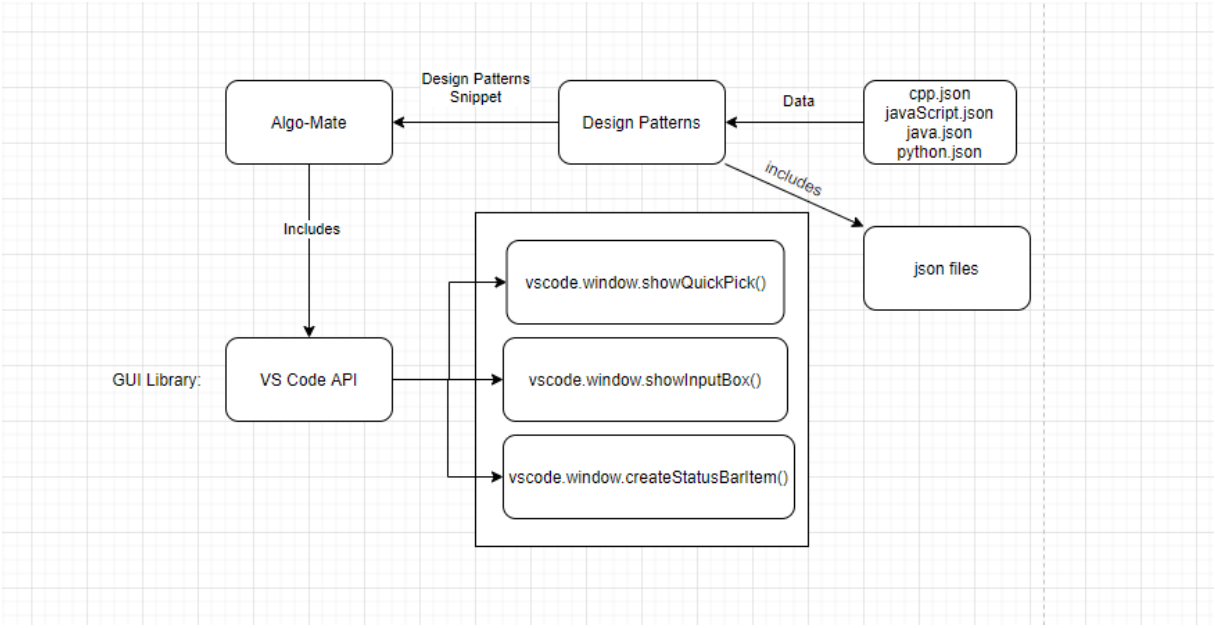

Figure 1. Use Case Diagram - Selecting Design Patterns with Algo-Mate

After determining the correct design pattern to autofill (see Figure 2), the respective design pattern is placed into the file you are working on, containing comments on how the pattern works. These comments detail the operation of the design pattern and point the user in the right direction when choosing what variables to make edits to in order for the design pattern to be effectively implemented into their project (see Figure 3). Algo-Mate gives you the option to step through the design pattern, changing key variables and values to update the template for your use case.

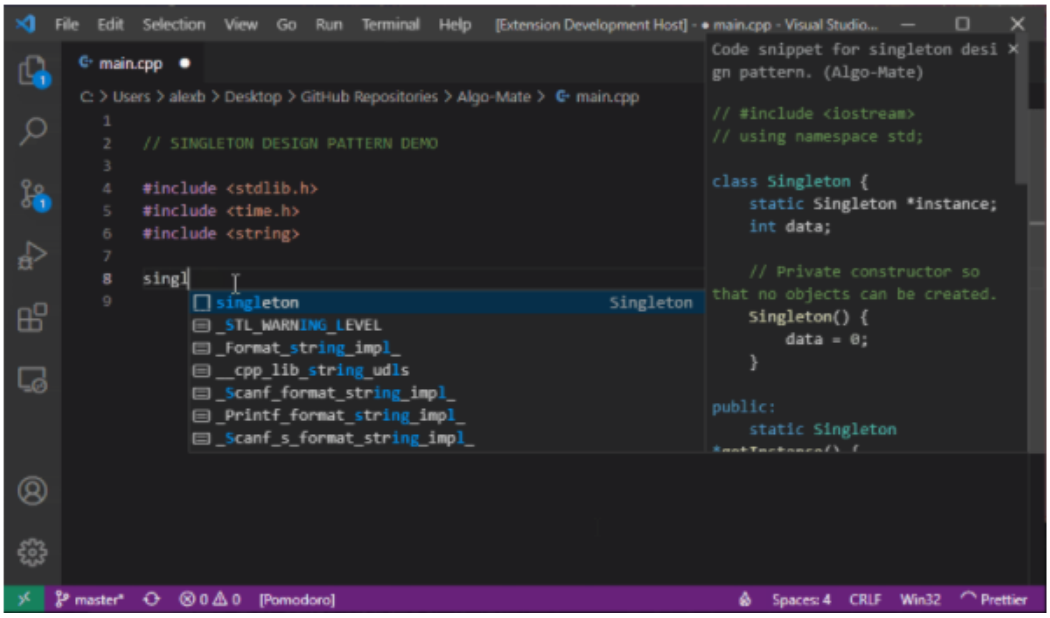

Figure 2. Design Pattern Auto-Completion: Singleton

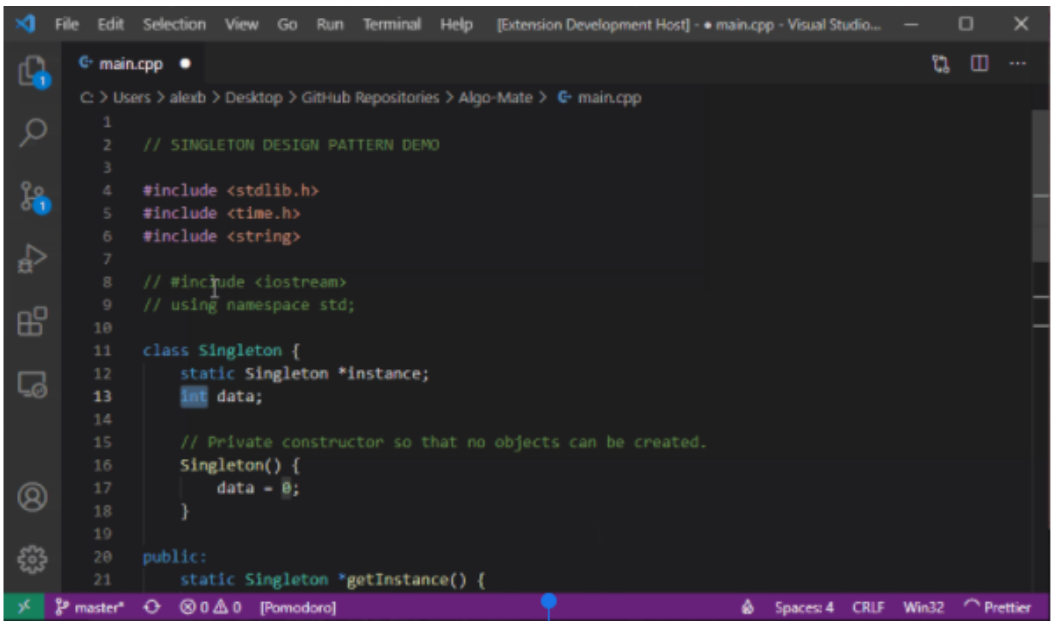

Figure 3. Singleton Design Pattern Template

Alternatively, this can be accomplished through the Algo-Mate UI, located at the bottom of the status bar in VS Code (see Figure 4 & Figure 5 below). By clicking the Algo-Mate option in the status bar, we can select a design pattern and the language we wish for it to be implemented in, and insert it into the file we are currently working within.

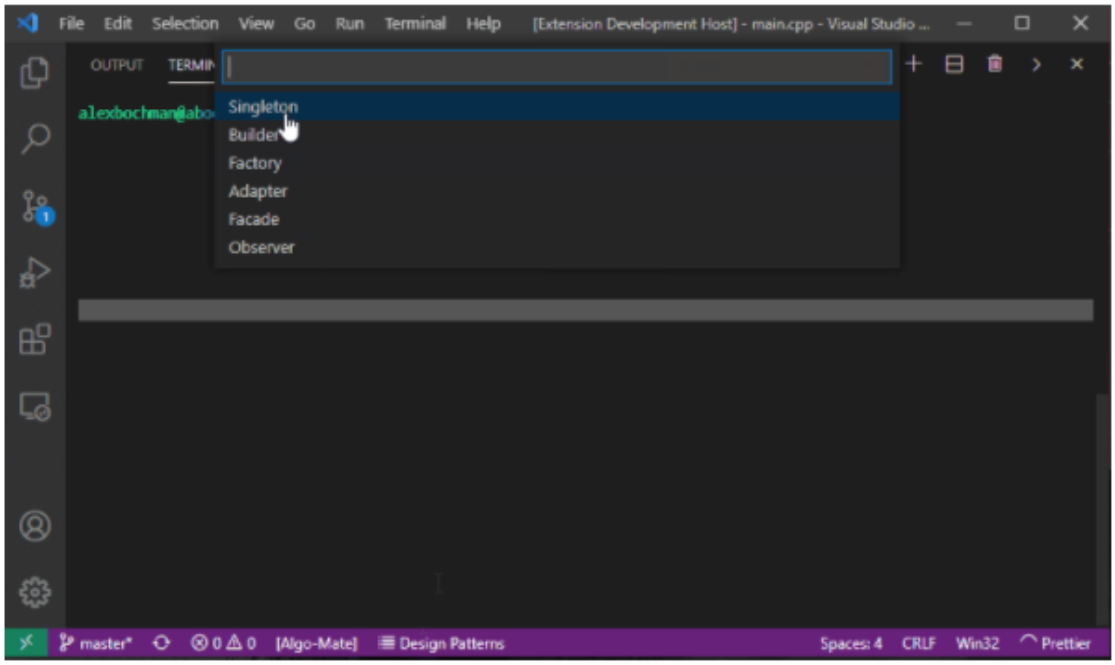

Figure 4. GUI Design Pattern Selection Options

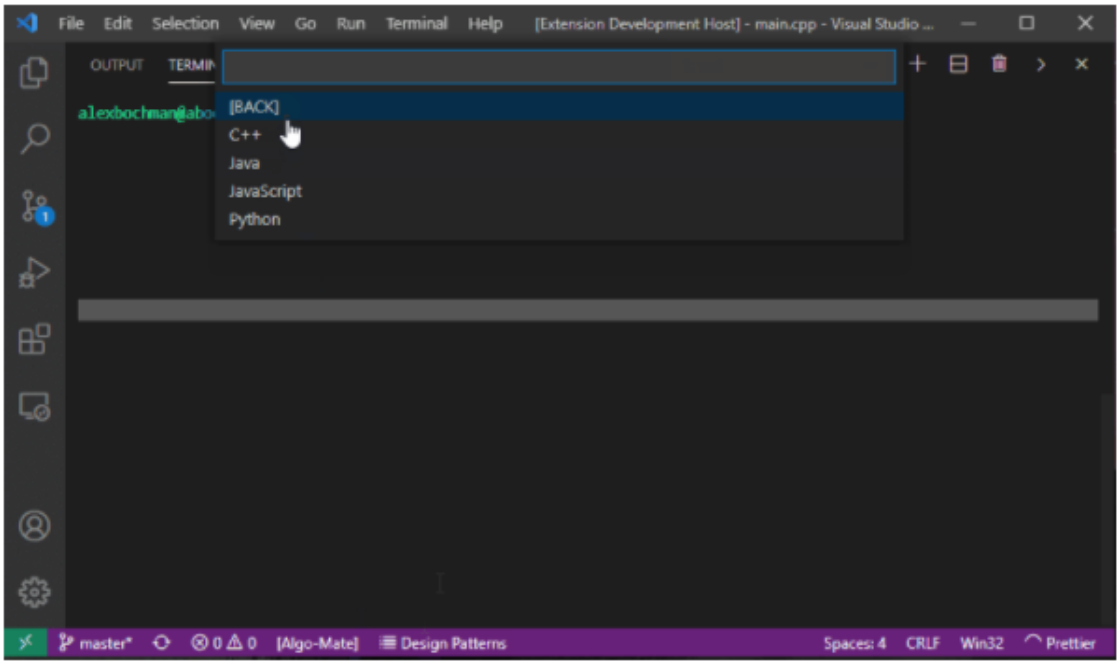

Figure 5. GUI Languages Selection Options

The interactions are depicted in Figure 6 below, where a user opens VS Code, and begins to type in a design pattern, letting Algo-Mate autocomplete the design pattern or by selecting the design pattern through the GUI. We saw this be completed in Figures 2 & 3 for typing the design pattern, or in Figures 4 & 5 for going through the GUI.

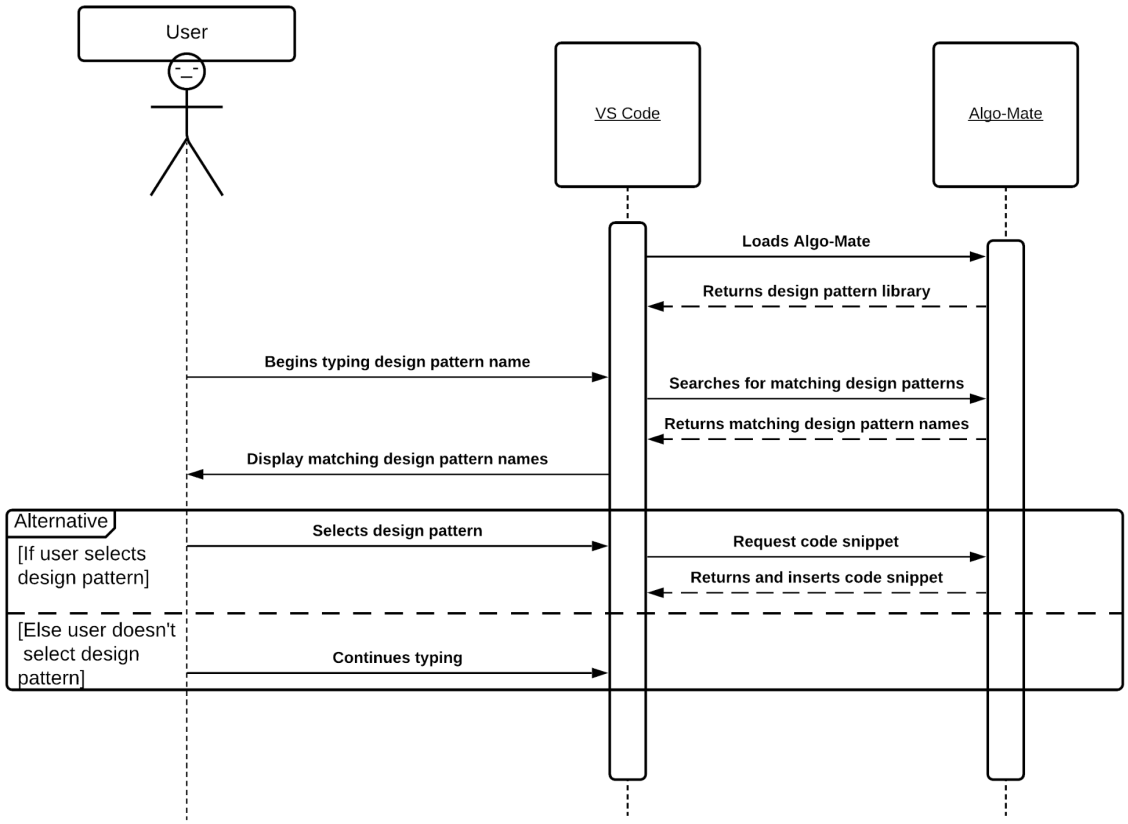

Figure 6. Design Pattern Sequence Diagram - Selecting Design Patterns

Currently, something like this does not exist within the VS Code marketplace, leaving a huge opening for Algo-Mate to fill. Algo-Mate improves upon the foundations of IntelliSense, while adding robust functionality for design patterns in the process. With the addition of comments to the template, Algo-Mate aims to remove the friction of learning design pattern templates, and allows users to get right into implementing them within their projects. With a deeper understanding of the design patterns provided, we hope users will be able to broaden their techniques without spending hours of time researching exactly how to implement each design pattern from scratch. If this research is needed when implementing the design pattern, Algo-Mate also provides links to resources that can be used to gain a deeper understanding of the implementation of the design pattern. This allows users to have reputable resources to go to when looking for information, allowing them to have resources to go to without wasting time trying to find documentation.

Building and Testing

Building Algo-Mate can easily be accomplished by following the instructions[4] below:

Download and Use Extension

1. Make sure Visual Studio Code is installed. Download here: https://code.visualstudio.com/download
2. Download and install the extension
    a. Option 1: From Visual Studio Marketplace: https://marketplace.visualstudio.com/items?itemName=AlexBochman.algo-mate
    b. Option 2: From within VS Code, click on the extensions tab on the left-hand sidebar and search for "Algo-Mate"

Clone, Build, Develop, and Test Instructions

1. Make sure Visual Studio Code is installed. Download here: https://marketplace.visualstudio.com/items?itemName=AlexBochman.algo-mate
2. Clone the latest version of Algo-Mate: https://github.com/alexbochman/Algo-Mate.git
3. Open the Algo-Mate project in VS Code
    a. Make any changes to the files in the "src" folder, or the package.json
4. Click Run > Start Debugging to test Algo-Mate
5. In the VS Code development window that pops up
    a. You'll see [Algo-Mate] on the status bar. Clicking it will bring up UI options.
    b. Typing out any of the prefixes from any of the JSON files within a file that qualifies (.cpp, .java, .js, .py) will display design patterns that can be auto-completed.
    c. After auto-completing a design pattern, points of interest (POI) will be highlighted. They can be modified, and groups of POIs will be changed automatically while still highlighted.
    d. Hitting tab will navigate to the next POI.
    e. Hitting shift+tab will navigate back to the previous POI.

Algo-Mate has Continuous Integration (CI) being accomplished through Travis CI. It is linked to the GitHub repository and runs testing every time a commit is pushed to GitHub. These test cases were built out as functionality was added to the project. All elements of the project are being tested for functionality and efficiency, all of which are being shown on the build history located on Travis CI[5]. The steps we took to implement continuous integration are listed below:

1. Created a YAML file (.travis.yml) to automate building and testing.
2. Configured Travis CI to work with the GitHub repository.

---

[4] https://github.com/alexbochman/Algo-Mate/blob/master/README.md
[5] https://travis-ci.com/github/alexbochman/Algo-Mate/builds

3. Wrote basic tests that failed and succeeded when expected.
4. Wrote tests that implemented VS Code extension API. This is currently an issue as our testing frameworks don't recognize VS Code API, causing errors, and causing our builds to fail. More research and troubleshooting with JavaScript test frameworks will be conducted to alleviate this issue.

Results

The original proposal to implement auto-completed algorithms was lacking in novelty and usefulness, so we've pivoted and came up with a plan to develop an extension centering around design patterns. See the extensive results list below:

- Implemented auto-completion for the singleton design pattern for C++ within our cpp.json file. Additionally, this allows users to tab back and forth to different points of interest (POI).
- Published the extension to the VS Code extension store using node package manager.
- Tested autocompletion between different file types for the supported languages (C++, Java, JavaScript, Python). For example, autocompleting the singleton design pattern will check what the user's current file type is, and will pull the design pattern from the correct JSON file. In short, you won't see JavaScript code in a C++ file. This is due to configurations within our package.json file.
- Published a web page to compile all of the extension's information and documentation.

Ensuring Algo-Mate is inserting design patterns swiftly is key to a user's experience. For this reason, we want to ensure the time to insert a design pattern of a certain language is consistent, and does not take long. To ensure Algo-Mate is performing as intended we have created an automated script to run insertions of a design pattern and records the time it takes to complete this operation. This script is located within the "user-interface.js" file, where once the Algo-Mate application is activated, the function getBuildData will run automatically. It will run 100 insertions of a design pattern, recording the time (in milliseconds) it takes to complete the operation. It then converts this data to a csv called "data.csv" and exports it to the users workspace.

This data allows us to see variations in the insertion times, identifying where issues may be arising. After this data is gathered we can generate a histogram for each design pattern and see the range of insertion times for each design pattern. An example can be seen in Figure 7 below, where we take 100 runs of the Singleton design pattern insertion and generate a histogram showing the times it took to complete this operation. Please note, insertion times may differ from machine to machine, so your results may differ from this example. This is common and is nothing to be concerned about.

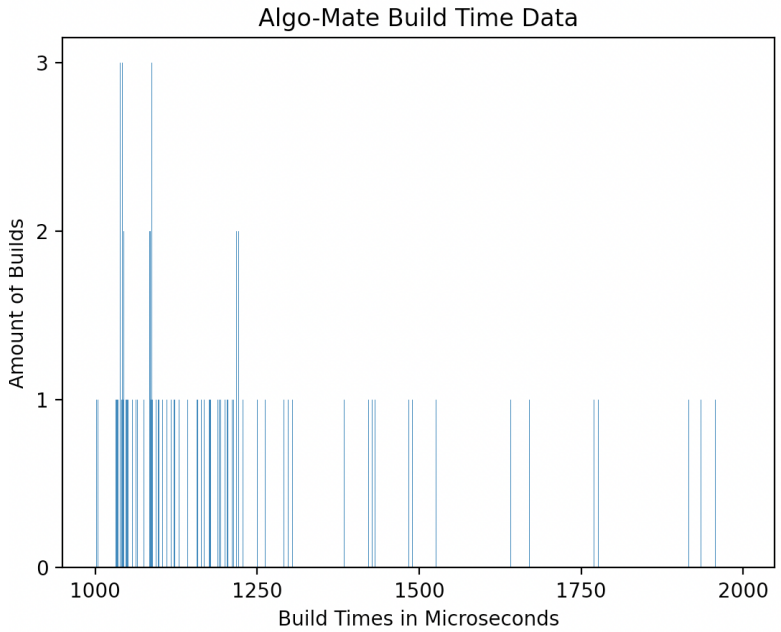

Figure 7. Average Insertion Time (in Microseconds) For Singleton Design Pattern

This figure can be generated by running the "Algo_Mate_Results.py" file using the following command "python3 Algo_Mate_Results.py data.csv" where the csv file we are using is provided as a command-line argument. This csv is generated beforehand using the "user-interface.js" file, as mentioned above. More information on running this can be found within Algo-Mate's README.

Deliverables

By the end of the project, Algo-Mate has been fully implemented as a Visual Studio Code extension which can insert code fragments of design patterns with comments to help the user implement them easily into their code. This plugin has been built out for C++, Java, JavaScript, and Python. The Algo-Mate extension itself has been built using JavaScript, and is capable of detecting what language the current file is written in, allowing for the correct design pattern to be placed in without the user selecting which language they are using manually when using the auto-completion feature.

For each language, four total design patterns have been implemented in a JSON file which are supported by Algo-Mate, these being: Singleton, Builder, Adapter, and Observer. These will cover the three types of design patterns, that being creational, structural, and behavioral. Included in the extension is a README file to help users install and quickly begin using the extension. This README contains instructions on how to select design patterns, choose the right language, edit the templates, and more.

Risks

The risk factors for the Algo-Mate project involved certain conflicts with the structural, behavioral, and creational design patterns. Creational patterns such as the Singleton pattern, may have been difficult to complete successful unit tests on because most testing frameworks rely on inheritance when it mocks objects. This could have been a problem because the singleton class is private and having to override static methods is usually impossible in most languages. The Adapter Structural Pattern also presented problems due to its overall complexity. If the user were to implement the Adapter Pattern, they may need to create new interfaces and classes that match the rest of their code. To deal with this, we've provided comments and tips within the design patterns to help the user understand how it works so they can integrate it well enough with their code. It was important that we had all design patterns be as straightforward as possible in order to prevent contingencies for the user.

Originally, the issue we expected to face was implementing several languages before the deadline. Fortunately, we ended up completing the design patterns for C++, Java, Python, and JavaScript. Since C++ and JavaScript were the main two languages we were all comfortable with, we implemented the design patterns in those languages first. Prior to implementing the patterns in Python and Java, we had to review the basics of those languages which took up some of our time.

Many of the challenges we've encountered while developing Algo-Mate have been due to VS Code's API. While we were mostly successful developing tests and implementing continuous integration, we struggled with choosing a JavaScript test framework that was compatible with VS Code modules during the build and test phase. Another risk we encountered was creating these design patterns in Java. Since Java doesn't allow for more than one public class per file, creating several public classes in one file was impossible. This issue arose when we worked with the Builder Pattern, which works efficiently when it has multiple class files and builds them together in return.

The main payoff for this project is that users will have access to a list of common design patterns that can be used with their code rather than having to look up the algorithm's basic structure. By doing this we learned more about the different design patterns ourselves and we look forward to spreading our knowledge to the user.

Discussion

The original proposal of Algo-Mate was to autofill completed algorithms instead of design patterns. As a team we quickly realized this was not going to be as impactful as we initially planned for, causing us to shift focus to a more defined topic, this being design patterns. Many libraries for different programming languages will already allow for the insertion of algorithms either by function calls or by a similar IntelliSense insertion as we were trying to accomplish. This meant we were essentially planning to make a tool that was already mostly implemented within the programming languages themselves. The team then shifted the focus of

the project to inserting design patterns instead, which are not as widely supported for insertion through other means. Algo-Mate would also provide a way to read up on these design patterns through reputable sources, allowing for anyone trying to implement these patterns in their projects to read up on how they work and where they can be used. Since the scope of the project shifted after getting started, not as many design patterns were implemented as we had hoped for. The team has implemented four design patterns in four different programming languages, allowing users to have a wider range of possible use cases within their own projects. Working on this project has helped the team develop an understanding for what users are looking for in software applications, helping the team develop good tactics for analyzing the use case of a project to determine its possible impact for the potential user base. We hope that what the team has made can help users for years to come, introducing more people to design patterns and reducing the friction of learning and implementing them in their own code. Through the use of the insertions and provided documentation, we believe Algo-Mate has accomplished its goal to make using design patterns easier for everyone.

Appendix A. Feedback

Throughout the course of the project the scope was changed drastically from focusing on algorithm insertion to design pattern insertion and assistance. As most of the feedback at the beginning of the project pertained to Algo-Mate's use case being too niche, the project was shifted. Below is a list of relevant feedback that was addresses throughout development, ensuring the project was well developed into a full functional and impactful tool:

- Addressed concerns with Algo-Mate applying to too niche of an audience with just algorithms. The project was shifted to a focus on design patterns to allow for a broader, more defined use case.
- Added elements to allow the user to obtain relevant information on the design pattern they are trying to implement from reputable sources. This helps prevent the user from having to spend excess time trying to find the right documentation about a particular design pattern and instead go right to a resource that can help them immediately.
- Adjusted the potential risks the user may face now they're focusing our project towards different design patterns instead of generic algorithms.
- Fixed the use-case diagram in Figure 1 so it addresses its structural design and background process.
- Adjusted the sequence diagram in Figure 2 as a triggered event instead of a loop and showed how the user interacts with Algo-Mate.
- Addressed issues of explaining UI elements and not showing them in use.
- Addressed feedback of having the python script plotting build time data to auto export the figure as a file.
- Addressed issues where the data gathered was displayed in ms which grouped the runtimes too tightly. The team is currently determining a way to display the runtimes in µs.
- Added an appendix to store information important to the timeline and development plans for the project.
- Addressed issue where CI information was under the *Results* section. The CI information has now been moved under the *Building and Testing* section.
- Addressed issue where the project links was in an awkward spot under the *Results* section. *Project Links* is now its own section at the end of the document.
- Addressed the risks that we planned on encountering prior to starting this project and included new risks that we encountered along the way.

Appendix B. Milestones

The first major milestone which was completed during the mid-semester week had to be adjusted as our implementation plans changed. Initially by March 5th, 2021, we planned to have a functional webpage on GitHub with appropriate tabs for the project, four implementations of algorithms to be auto-completed and a barebones functionality of the extension is published to the VS Marketplace. Due to our shifting towards implementing completion for design patterns, we were only able to implement one completion: Singleton pattern, however we met all other goals that we set. Having a functional webpage operational early will allow the team to use it as a progress tracker. As we continue the project, new deliverables and release notes will be published regularly as the semester progresses. When the semester comes to pass, the webpage will be completed and will act as an accurate record for anyone interested in the Algo-Mate project.

The other major component of Algo-Mate is the UI. Buttons and options have also already been implemented as part of our second milestone to allow the user to control certain parts of Algo-Mate and use the help menu to find information on the design patterns they wish to insert. The UI was completed by March 26th, 2021 as planned. Auto-completion of four design patterns: Singleton, Builder, Adapter, and Observer will be implemented in C++, Java, JavaScript, and Python by April 19th, 2021.

As we approach the final major milestone, we will complete the final bug fixes, wrapping up the code by April 21st. The demonstration video and final presentation will be completed by April 24th. This will allow for any small edits to be made before the final week of classes, which takes place starting April 26th, 2021. Completing the project before this week is essential because the demonstration video, class presentation, and final iteration of the project web page depend on it. The Algo-Mate team will spend the weekend of April 24th & 25th rehearsing the presentation before presentations begin the following week.

Appendix C. Schedule

Date	**Goal**
February 18th, 2021	Proposal Completion – Compile each component of the proposal that was divided among the members
February 19th, 2021	Proposal Due Date – Proofread and submit proposal before class
February 26th, 2021	Work on auto-completion of design patterns in C++
March 5th, 2021 Mid-semester week	• Have a working webpage up this week with these sections: overview, members, updates, and a link to GitHub

	• Complete implementation of auto-completion for four design patterns in C++. • Publish this version on the Visual Studio Marketplace
March 26th, 2021	Finish UI
April 19th, 2021	Finish auto-completion of four design patterns in each of our four languages: C++, JavaScript, Java, and Python
April 21st, 2021	Finalize bug fixes, ensure code is in a polished and functional state
April 24th, 2021	Complete the final presentation and begin rehearsing
April 26th-April 30th, 2021 **Final week of classes**	• Completion of the webpage • Publish completed extension on Visual Studio Marketplace • Complete and submit final reports and demonstration video • Present completed project to the class during this week

Project Links

- GitHub repository: https://github.com/alexbochman/Algo-Mate
- Algo-Mate extension download for VS Code: https://marketplace.visualstudio.com/items?itemName=AlexBochman.algo-mate
- Project webpage: https://alexbochman.github.io/ClassesAndProjects/SW-Eng/Algo-MateProject/Algo-Mate.html#