A Tool to Visualize the Structure of a Codebase Using Information Foraging Theory Design Patterns

Final Report | Team Postal | Group #38

Cramer Smith, Sam Lichlyter, Eric Winkler, Zach Schneider

Abstract: Developer tools are often complex pieces of software. Gathering and manipulating useful information for a programmer can often be a slow and costly process. By implementing Information Foraging Theory design patterns in the creation of these tools, the information collected may be more useful or produced faster. Information Foraging Theory is the theory and math behind the choices people make to maximize the value of the information they find versus the cost of getting that information. The aim of this project is to develop a tool that will act as a proof of concept to this idea and increase developer efficiency. Through the implementation of multiple IFT design patterns, the Postal team will create a developer tool that helps enforce and maintain code structure.

CONTENTS

I. Introduction

A. Client and Project Origins

This project originated in August 2016 when one of the the team's members, Sam Lichlyter, was made aware of a research opportunity offered by Professor Christopher Scaffidi from Oregon State University. Prof. Scaffidi's primary research area is in Information Foraging Theory (IFT) which revolves around studying how humans find and utilize information sources. Sam, already a student researcher under Prof. Scaffidi, asked if he could create a Senior Capstone project out of IFT in a new study concerning software developers. Prof. Scaffidi prompted Sam to identify a team and to come up with a project proposal that would develop a software tool according to the principles of IFT. Sam included Eric Winkler, Zach Schneider and Cramer Smith in the potential Capstone team. There were two proposals submitted for consideration: an interface to allow developers to read and search program log files more easily, and an extension for an integrated development environment (IDE) that would automatically organize a developer's code into appropriate concerns i.e., the data layer, interface layer, and application layer. The latter of the proposals was selected and this Capstone team was formed to undertake it.

B. Project Purpose and Expectations

A more detailed explanation of what the team was tasked with doing is as follows. First, design a tool according to IFT design patterns that will help developers find and utilize information, in this case, within an IDE. Second, build and code said tool until it is in a state where it is useful in its main purposes. Next, organize a series of formal user tests to determine if the tool aides developers in performing a set of real world tasks. Finally, analyze the results of the tests and, if the results are positive, write a formal research paper describing the entire process and its relationship to IFT. Prof. Scaffidi supervised the design process in the Fall. The team met with him about once and month to check in and discuss any design questions or changes. Emails were exchanged a bit more often. He left the coding and implementation phase largely up to the team during the end of the Fall through the Winter and contact was infrequent. Spring term saw contact pick up again as the team finalized the product with him and began arranging the necessary materials for user testing. Prof. Scaffidi has and will continue to directly facilitate the actual testing and analysis process until it is complete.

C. Project Roles

Throughout the year, each team member assumed and carried out various roles depending on the stage of the project. Every member generally contributed to all aspects of the project, but the responsibility for some areas was assumed primarily by one individual.

Sam served as the team's primary client contact since the relationship had existed prior to the beginning of this project. He had a major role in the design and implementation of the parser component of this tool, along with Eric. Additionally, Sam wrote many of the grammars and rules that the parser would use to find specific data within developers' codebases.

Zach took primary responsibility for project documentation, deadlines and general administrative work. He contributed to early iterations of the data structure used to store parsed data within the tool. He then transitioned to adding features to the user interface (UI) and testing various use cases within it.

Cramer did much of the research into which IDE the tool would be developed for. Once Visual Studio Code was decided upon by the team as the platform for this project, he also created the core of code required to launch and include extensions within that platform. Cramer deployed each iteration of the tool to the Microsoft VS Code Extension Gallery online when various project milestones were reached.

Eric served as overall project architect, coming up with the general structure of the project from the original proposal and changing it as necessary throughout the year. He worked with Sam in building the parser and created the core of the UI and Electron window code. When the parser was modified in the middle of the year, he also re-worked the data structure system being used.

II. Original Requirements and Timeline

The original Requirements Document for this project is inserted in the following pages.

A Tool to Automatically Organize the Structure of a Codebase Using Information Foraging Theory Design Patterns

Requirements Specifications Document

Team Postal

Cramer Smith, Sam Lichlyter, Eric Winkler, Zach Schneider

November 4, 2016

Abstract: Developer tools are often complex pieces of software. Gathering and manipulating useful information for a programmer can often be a slow and costly process. By implementing Information Foraging Theory design patterns in the creation of these tools, the information collected may be more useful or produced faster. Information Foraging Theory is the theory and math behind the choices people make to maximize the value of the information they find versus the cost of getting that information. The aim of this project is to develop a tool that will act as a proof of concept to this idea and increase developer efficiency. Through the implementation of multiple IFT design patterns, the Postal team will create a developer tool that helps enforce and maintain code structure.

CONTENTS

I. Introduction

The Postal extension is being made with two goals in mind. The first is to provide evidence that the use of IFT Design Patterns in the production of developer tools is beneficial. The second is to create a tool that will help developers write and maintain clean code bases for websites. The Postal extension will do this by giving the user reminders and suggestions about the best coding practices of the current language that they are using and by providing a visual representation of their project's code base.

A. Purpose

The purpose of the Software Requirements Specification of the Postal extension is to describe the extension's functionality and usage for the target user. This document will also help Team Postal and its stakeholders have a better understanding of the project's purpose and goals.

B. Scope

The Postal extension will be developed by Research Experience Undergraduates lead by Dr. Christopher Scaffidi. The project will be an extension for Visual Studio Code. The purpose of this extension is to help new programmers develop good practices in their styling of code. The extension will be aimed at new developers so that it will help them avoid mistakes made during web development. This extension will be able to parse the code that the user is editing and will be able to give them suggestions based on the current W3C standards.

C. Definitions, Acronyms, and Abbreviations

Information Foraging Theory (IFT): An approach to the analysis of human activities involving information access technologies. The theory derives from optimal foraging theory in biology and anthropology, which analyzes the adaptive value of food-foraging strategies.[1]

IFT Design Patterns: General, reusable solutions to common design problems.[2]

Visualize Topology: To reveal the structure of the information topology to help developers more easily navigate structural relationships, backtrack, and make better decisions about which patches to visit.[2]

Notifier: Automatically notify the developer of a change in an information patch which may result in prey desired by the developer appearing in the patch.[2]

Dashboard: Generate an information patch in which a developer can become aware of links that lead to continually changing information patches relevant to his or her work.[2]

Gather Together: Enable a developer to assemble information features from disparate patches into a single patch, thus reducing the cost of navigation between those features.[2]

Reduce Duplicate Information: Enable developers to quickly forage by reducing the size of the topology. Here, the size of the topology is reduced by eliminating nodes with duplicate information.[2]

VSC, VS Code: Visual Studio Code.

IRB: Institutional Review Board.

File Map: The graphical user interface for visualizing project files and links, as well as errors inside code.

Error List: A list which displays all errors that the parser detected during its last run.

World Wide Web Consortium (W3C): The World Wide Web Consortium is an international community where Member organizations, a full-time staff, and the public work together to develop Web standards. [3]

Bibliography

[1] P. Pirolli and S. Card. (1995) Information Foraging in Information Access Environments. Xerox Palo Alto Research Center. [Online]. Available: http://delivery.acm.org/10.1145/230000/223911/p51-pirolli.pdf

[2] T. Nabi, T. Sweeney, S. Lichlyter, D. Piorkowski, C. Scaffidi, M. Burnett, and S. Fleming. (2016) Information Foraging Theory. Oregon State University. [Online]. Available: http://research.engr.oregonstate.edu/ift/readonly.php

[3] World Wide Web Consortium. (2016) About W3C. World Wide Web Consortium. [Online]. Available: https://www.w3.org/Consortium/

D. Overview

The remainder of this document will be concerned with the detailed requirements of Project Postal. It will also describe what the Postal extension is specifically and how it will be used by developers.

II. Overall Description

Postal is an extension for Visual Studio code that will help entry-level developers while they make websites using HTML, CSS and JavaScript. The extension will parse the code they write and will assist in identifying errors or bad coding practice.

A. Product Perspective

The product is aimed toward individuals that are learning web development. This product aims for users to find that the tool makes the development process more educational and easier to understand. This checking tool differ from similar code checkers in that it will be offered in a free, cross-platform, light weight text editor. This will make the correction and error identification more accessible to the users.

After running, the generated extension should have the following structure:

```
.
├── .gitignore
├── .vscode                     // VS Code integration
│   ├── launch.json
│   ├── settings.json
│   └── tasks.json
├── .vscodeignore
├── README.md
├── src                         // sources
│   └── extension.ts            // extension.js, in case of JavaScript extension
├── test                        // tests folder
│   ├── extension.test.ts       // extension.test.js, in case of JavaScript extension
│   └── index.ts                // index.js, in case of JavaScript extension
├── node_modules
│   ├── vscode                  // language services
│   └── typescript              // compiler for typescript (TypeScript only)
├── out                         // compilation output (TypeScript only)
│   ├── src
│   |   ├── extension.js
│   |   └── extension.js.map
│   └── test
│       ├── extension.test.js
│       ├── extension.test.js.map
│       ├── index.js
│       └── index.js.map
├── package.json                // extension's manifest
├── tsconfig.json               // jsconfig.json, in case of JavaScript extension
├── typings                     // type definition files
│   ├── node.d.ts               // link to Node.js APIs
│   └── vscode-typings.d.ts     // link to VS Code APIs
└── vsc-extension-quickstart.md // extension development quick start
```

Figure 1. Basic file structure for a VS Code extension

1) System Interfaces: Visual Studio Code was chosen as the sole system interface for Postal. VSC is free, lightweight, cross-platform and open source, making it accessible to both developers and users of this development tool.

2) User Interfaces: The main interface that Team Postal plans to implement is a graphical representation of the user's files. Users will be able to quickly navigate through a visual representation of their files in the objective that this will give new developers a different way of thinking about their projects. This different perspective will ideally encourage the user to implement better file structure and organization in their projects.

3) Software Interfaces: Development of this project requires interfacing with the VSC extension API. Development will utilize the Node Package Manager tool that is included in Node.js for initialization of the extension interface. The project will also use a code generator called yeoman to generate skeleton files which will edited and made into an extension using JavaScript. The skeleton code will be the base of what the rest of the extension is built on.

4) Communication Interfaces: The only communication interface that Postal will require is an ability to download the extension from the VS Code Extension Marketplace.

5) Operations: Postal will have two main modes of operation. The first being the active parsing, and the second being the file map mode. The parsing mode is an unattended operation. As the user saves their files, the parser will parse and analyze the text checking for errors or malpractices as defined by W3C. The second mode of operation

an interactive mode. The user will utilize the visual file map to navigate to files, to visualize the file structure, and to locate the errors that the extension specifies within the code. All of these functions are partly data processing and support functions. The main goal of the extension is to process data using IFT design patterns.

B. Product Functions

From a functional perspective, Postal will largely do two things: Provide the user with a visual perspective of their project folder and parse HTML, CSS and JavaScript files for bad coding practices and information to feed into the visual interface. The visual interface component will be a separate window referred to as the 'File Map'. This window will scan the directory of the currently open project and create a visual representation of the files in the folder. The representation will also feature indicators of errors that the parser detects in files and visually display any links between files. The parser will read the HTML, CSS and JavaScript files in the user's project folder and look for specific violations of rules the defined in the system. These rules will be based on the W3C best practices guidelines. Additionally, the parser will ignore any lines of code that marked in an exception list, allowing the user to break rules in the case that they have to. Along with our product completing the aforementioned requirements, it also needs to be implemented using the IFT Design Patterns also mentioned previously in this document. The extension will be able to parse and interpret code projects with up to one hundred thousand lines of code quickly without noticeable lag. More specifically the extension will be able to handle files of this size with less than one second of lag. The entire extension should run seamlessly with the user taking little to no notice that our processes are running. The user should experience little to no lag while our extension is running depending on the power of their personal machine. This project will assume that an average user is using a computer with at least two gigabytes of ram. The project inherently won't have any safety requirements.

C. User Characteristics

Users of Postal will be web developers and computer science students with some level of prior development experience, though prior knowledge will not be required. The users will be familiar with the concepts surrounding programming languages and markup languages, such as HTML, CSS and JavaScript. Users will be familiar with file systems, such that the visualization within Postal will enhance understanding of project structure. Users should have a basic understanding of debugging techniques that will be aided by the error recognition and highlighting within Postal.

D. Constraints

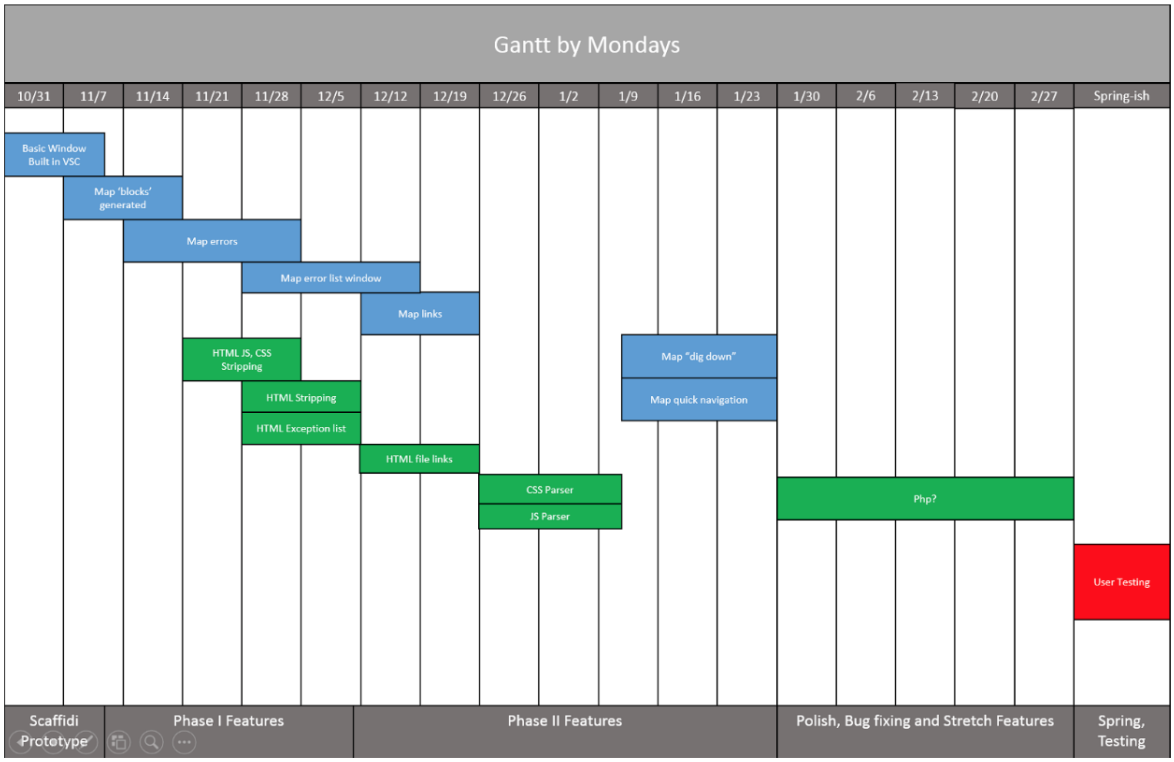
The Postal extension for VSC will be constrained chiefly by software and programming language level barriers. As this extension will be primarily written in JavaScript (ECMAScript2016 standard, specifically), it will be bound by any functional or design limitations that exist within that programming language. Additionally, all Postal functionality must be possible within the VSC environment. VSC does not detail any notable performance constraints in its documentation that would conflict with the goals of Postal at this time. VSC does specify that extensions are not allowed access to the underlying UI DOM. Finally, the file visualization and linking features of postal will be

reliant on read, write and execute permissions within the scope of project loaded into VSC at that time. If any user's operating system restricts Postal access to any of those file system features, that operating system will also be a constraint to proper function.

E. Assumptions and Dependencies

The successful development of Postal assumes that Microsoft will not update Visual Studio Code in such a manner that breaks the function of the Postal extension. It is also assumed that VSC will continue to support the operating systems used in development, testing, and utilization of Postal.

F. Apportionment of Requirements



III. Specific Requirements

A. External Interfaces

The primary interface of the extension will be a window within the Visual Studio Code environment. This window will contain two main elements (the File Map and the Error List) as well as a toolbar for options.

The File Map is a visualization of the user's project. When a project is opened within VSC, the extension will automatically populate the map by scanning the directory of the opened solution. This map will display each file found in the project directory in an organized fashion. The map will also feature options to display a visual indicator of links and calls between files in the directory (for example, if an HTML page has an image embedded in it, the link option would indicate a link between the HTML and image files) as well as an option to display an indicator

of the location of errors within the GUI.

The user will also be able to interact with the File Map by "digging down" into a file. This process will allow a user to click on a visualized file and have the map display more details about that file. For example, if an HTML file is clicked on within the map, the UI will update and visualize details of the file like divs and links as their own objects. If the above mentioned error option is enabled, the object containing the error would indicate that.

The Error list is a list which displays all errors that the parser detected during its last run. The user will be able to click on a particular error in this list which will then trigger the extension to navigate the user to that particular error in the code. Additionally, if the user hovers over a particular error, the corresponding location in the file map will be highlighted.

B. Functions

1) Parser Related Functionality:

- Functionality capable of parsing JavaScript, HTML and CSS documents. As parsing occurs, particular instances of code that break rules defined in the system will be flagged. Flagged lines will be visualized in the File Map.
- The system must have a method for storing rules that will be used by the parser to determine if a line of code should be marked as an error. Many of these rules will be based on W3C best practices for the particular parser.
- The system must have an editable list of exceptions. Exceptions are defined segments of code that may break a parsing rule, but because it was either intentional or necessary, should not throw an error.
- Current Parsing Rules for HTML:
  - Flag JavaScript and CSS code that is not in the exception list.
  - Make a note of each use of another file within the HTML file. This will be used to visualize links between files in the map.
- Current Parsing Rules for CSS:
  - Flag styles that can be optimized (ids vs. Classes).
  - Flag redundant definitions.
  - Do not flag code in the exception list.
- Current Parsing Rules for JS:
  - Flag Global variables that are not defined at the top of a file.

2) File Map Functionality:

- Display in a visual, chart-like manner all files in the project directory.
- Display links and calls between files. This option can be turned off by the user.
- Display indicators of errors (broken rules) at the corresponding location within the map. This option can be turned off by the user.

- The system must allow for the Dig Down functionality. When an object in the map is clicked on, the map will update to display the object in more detail. Details will often be their own object. Error indicators will be updated.
- Display an error list adjacent to the file map. Errors will be organized by location.
  - If the user hovers over a particular error in the list, the corresponding location in the file map will be highlighted.
  - If the user clicks on a particular error in the list, the extension will navigate the user to the location of the error in the code.

C. Performance Requirements

Our product will be able to support as many projects as each user has. Given a web site project consisting of less than thirty HTML files, five CSS files, and ten JavaScript files, our product should complete its analysis in less than a second. This example should be a pretty standard setup for most intermediate web developers.
Our parser should complete each HTML file in less than 1/5th of a second and each CSS and JavaScript file in less than 1/10th of a second.

1) Standards Compliance: Other than the standards set forth by the VSC extension API, there are no overarching standards our product needs to comply with. The plan is to test our product with people following the IRB protocols, and the testing will follow their safety requirements regarding the testers information.

D. Software System Attributes

1) Reliability: Postal should complete its analysis for 95% of the projects it is given. It should also satisfy the performance requirements 95% of the time.

2) Availability: The extension will always be available as long as it is accessible from the VS Code Extension Marketplace. It will not need to pull any data from the internet for it to function, so it should always be available.

3) Security: The Postal extension will only save data that the user defines in the settings of the application. The extension will know nothing about the individual user, and therefore will have nothing to be kept secure.

4) Maintainability: Postal will be implemented using Object Oriented Programming techniques which will aid in the maintainability of the software. This will mean if one section of the code is changed that is used in multiple places it will only need to be changed in once instead of each of those places.

5) Portability: The Postal extension will be written in JavaScript and will be able to be run on any computer that is able to run Visual Studio Code. The development will focus mainly on the Windows and Mac OSX operating systems, with minor testing in some variants of Linux.

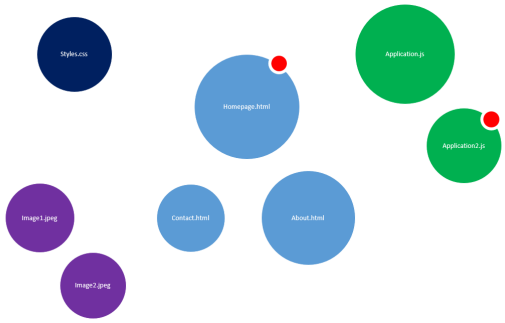


Figure 2. A mockup of what the file map could look like. In this example HTML pages are blue, CSS pages are navy, image files are purple and JavaScript files are green. Red dots indicate that the parser found a broken rule in the file.

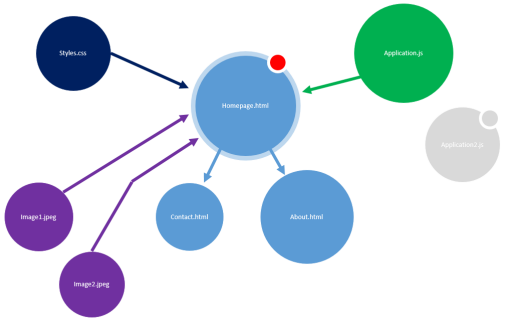


Figure 3. A mockup of the link feature in the file map. The Homepage.html file is 'selected' and all links to other files are shown via arrows. Note that files like Application2.js are greyed out as they do not have a link to the selected file.

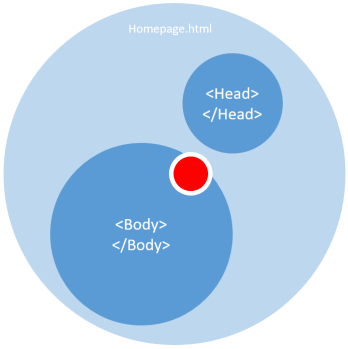


Figure 4. An example of what a 'dug into' Hompage.html file could look like. In this example, the visual is expanded to the next level down of HTML tags. These details may be expanded themselves.The error indicator from previous examples is also present and in this case states that the parse found an error in the Body section of the HTML document.

Client Signature

Signature

Date

Student Signatures

Signature

Signature

Signature

Signature

Date

III. Updates to Requirements and Timeline

Below is a simplified list of all requirements for this project as of the final Requirements document submitted on November 4th, 2016. No changes to requirements were made for the duration of the project.

A. Product Functions

0	Requirement	Changes	Comments
1	• Offered in a free, cross-platform text editor	• None	• Completed
2	• Provide the user with a visual perspective of their project	• None	• Completed
3	• Parse project for bad coding practice/incorrect formatting	• None	• Completed. This is implemented through the Notifications/Error grammars.
4	• Indicate to user when ërrorsäre parsed	• None	• Completed. Mid-winter we began to refer to errors by a more appropriate name: Notifications. The functionality remained unchanged.
5	• Visually display links between files	• None	• Completed
6	• Rules defined by user defined grammars	• In an earlier, pre-final version of the requirements documents, rules were defined by W3 best practices for web projects. This changed by the time of the final submission on November 4th to rules being defined by user grammars.	• Completed
7	• Ignore lines of code marked in an exception list	• None	• Completed. Implemented through comment grammars and the postal.ignore functionality.
8	• Parse projects up to 100,000 LOC with less than one second of lag	• None	• This requirement is situationally completed. It does work for projects of 100,000 lines but will have problems with individual files of that size.

B. External Interfaces

0	Requirement	Changes	Comments
1	• Two main elements: File Map UI and Error List	• None, some additionally functionality was added.	• Completed. Error List now refereed to as Notification List
2	• Toolbar for options	• None	• Completed
3	• Populate File Map with contents of opened solution	• None	• Completed. This is implemented through the Notifications/Error grammars.
4	• Visual indicator of links and errors	• None	• Completed
5	• ”Dig down” into a file to identify sub-nodes	• None	• Completed
6	• Error list displays all errors parsed	• None	• Completed
7	• User can click error to navigate to corresponding location in code	• None	• Completed, Implemented through double clicking a node
8	• User can hover over error and will highlight location in File Map	• None	• Completed

C. Functions

0	Requirement	Changes	Comments
1	• Be able to parse JavaScript, HTML and CSS	• None	• Completed, implemented through user grammars

D. Performance Requirements

0	Requirement	Changes	Comments
1	• 30 HTML, 5 CSS, 10 JS files in less than one second	• None	• Completed

E. Software System Attributes

0	Requirement	Changes	Comments
1	• Available on VS Code Extension Marketplace	• None	• Completed

F. Gantt of Implementation Progress for the Year

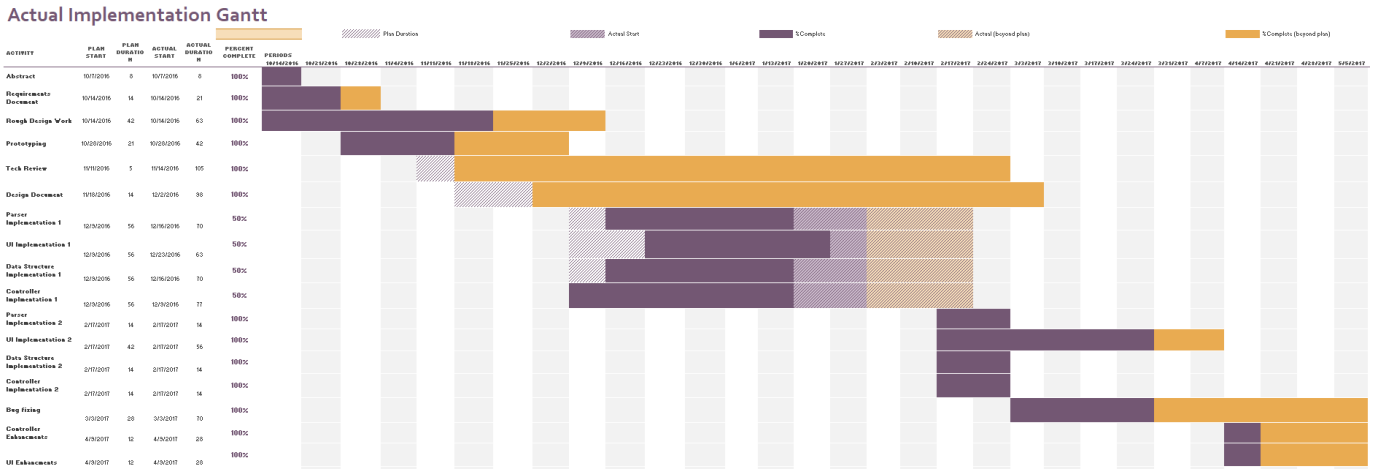


Figure 1. Gantt of the implementation progress for the academic year.

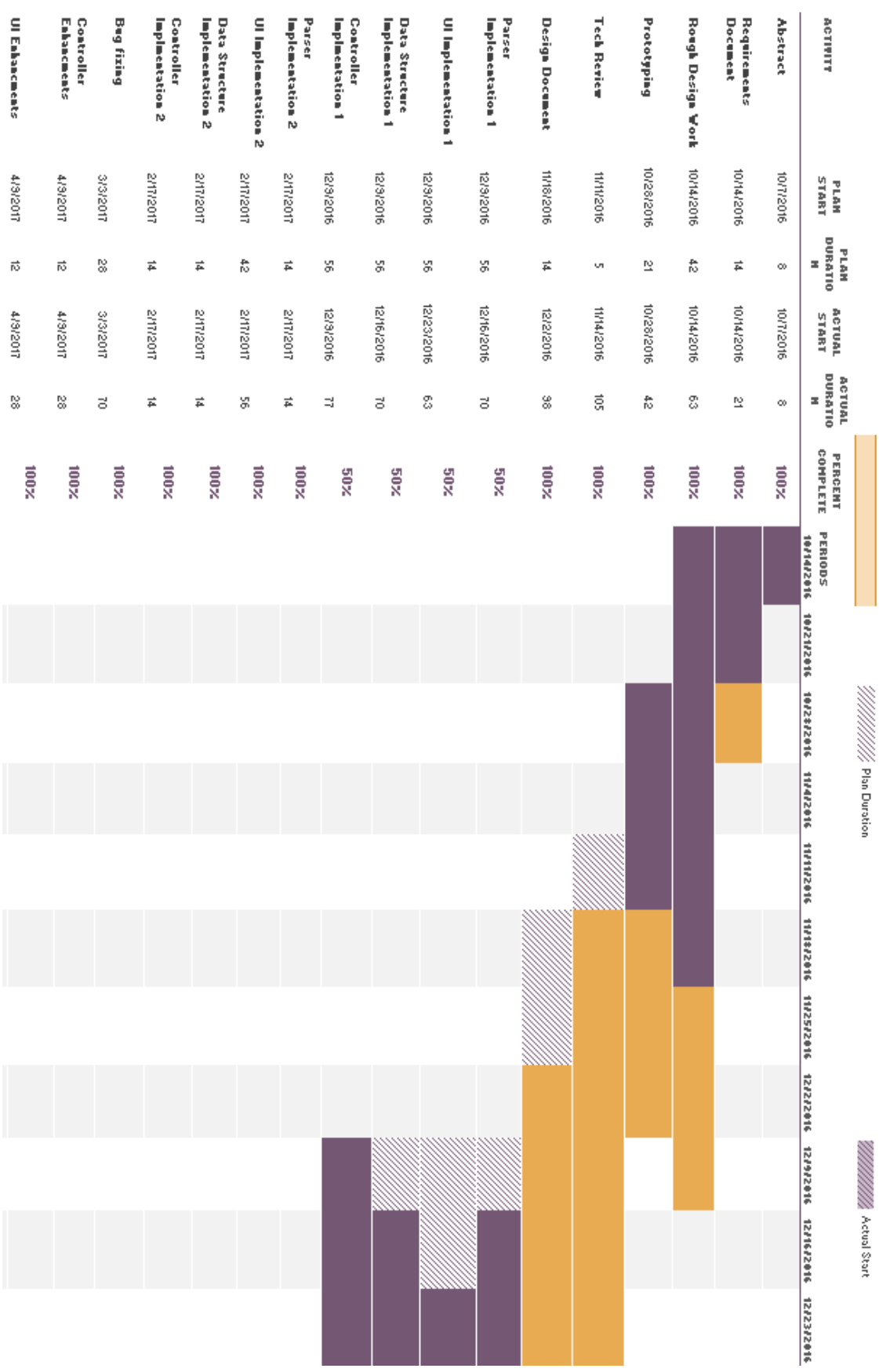


Figure 2. Top half of the Gantt for readability.

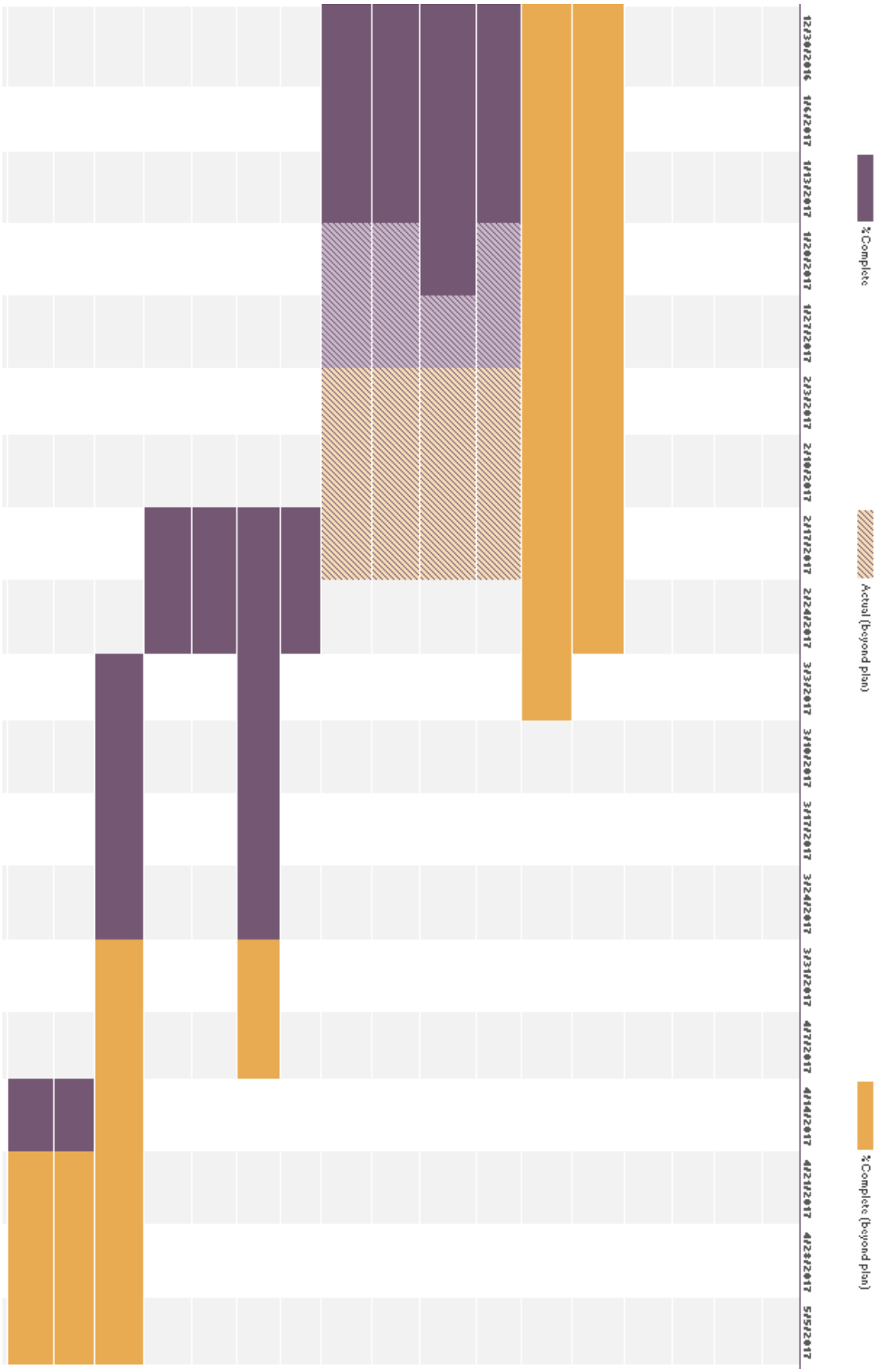


Figure 3. Bottom half of the Gantt for readability.

IV. Design Document and Changes

The original Design Document for this project is inserted in the following pages.

A Tool to Automatically Organize the Structure of a Codebase Using Information Foraging Theory Design Patterns

Design Document

Team Postal — Group #38

Cramer Smith, Sam Lichlyter, Eric Winkler, Zach Schneider

December 2, 2016

CONTENTS

A note for grading: Each major component and their subcomponents was covered by a different member of the team in this document.

- IDE/Extension – Cramer Smith
- Parser – Sam Lichlyter
- UI – Eric Winkler
- Data Handling – Zach Schneider

The majority of the design details on these components takes place in section 5 of this document. The preceding details and introductions were distributed amongst the team.

I. Overview

A. Scope

This document will cover the entirety design of the Postal extension written for the Visual Studio Code integrated development environment. The focus of the design will be on the four main parts of the extension, and the use of Information Foraging Theory within the extension. The four parts of the extension design are the parser, the data structure, the interface with Visual Studio Code and the user interface. The document with go through each of these parts and describe in detail how each will be implemented and how each part will function. The Information Foraging Theory Patters that are planned to be explored within the extension are the Specification Matcher, Structural Relatedness, Impact Location, Path Search, and Recollection. The document will go into more detail as to what these patterns mean and how they will influence the design of the extension.

B. Purpose

This design document describes the planned design and steps for implementing the Postal extension for Visual Studio Code. The team implementing the design will use this document as the blueprint for the implementation of the extension.

C. Intended Audience

This document is meant for the design stakeholders. The design stake holders include the team implementing the extension, their client, and the teams supervisors. The teams supervisors being the people grading the project on the implementation of the designs described within this document.

D. Conformance

This document conforms to the IEEE Std 1016-2009.

II. Definitions, Acronyms, and Abbreviations

A. Definitions

Model-View-Controller

A design pattern assigns objects in an application one of three roles: model, view, or controller. The pattern defines not only the roles objects play in the application, it defines the way objects communicate with each other. Each of the three types of objects is separated from the others by abstract boundaries and communicates with objects of the other types across those boundaries. The collection of objects of a certain MVC type in an application is sometimes referred to as a layer for example, model layer.[1]

Integrated Development Environment

A software application that provides comprehensive facilities to computer programmers for software development.

dictionary

An abstract data type composed of a collection of (key, value) pairs, such that each possible key appears at most once in the collection.

B. Acronyms

VSC

Visual Studio Code. Visual Studio Code is the IDE for which the Postal Extension is being built.

IDE

Integrated Development Environment.

UI

User Interface.

MVC

Model-View-Controller

III. Conceptual Model for Software Design Descriptions

This software will be loosely written with a model view control design pattern. It is loosely MVC because it is an extension and some of the view will be out of the control of the extension, but the main parts of the extension will fill these MVC roles. The model will be the data structure that will be used to represent the parsed files. The IDE and the user interface that the extension creates will be the view and be dependent on each other. The control will be the event handlers and the IDE, as these parts take and interpret the users actions that affect the data structure and in turn the UI.

A. Software Design in Context

The extension is designed to help novice web developer with organizing and create cleaner HTML and CSS code. To accomplish this, the design of the extension will contain a UI, a number of parsers for different languages, and a data structure to keep track of relevant information from the user's files.. The parsers and the extension's UI will need to communicate between each other and VSC. The communication between VSC and the parser will go though the data structure as VSC passes text to the parser and the parser populates information within the data structure.

IV. Design Description Information Content

A. Introduction

The following paragraphs will detail all primary pieces of this design document and the design of the Postal Extension. Information concerning who is responsible for the various facets of this piece of software, the major components of the software, the viewpoints from which these components will be designed, and the languages with which the components will be described will be detailed in the following sections.

B. SDD Identification

This document identifies the various components of the Postal VSCode Extension and how they function together. These components exist of the following:

- IDE

- Parser
- UI
- Data Handling

C. Design Stakeholders and Their Concerns

The design stakeholder for this design are the development team, the client, and the supervisors. The development team will be using this as a blueprint for the implementation of the postal extension. The client will use this to check that the development team is headed in the right direction as to what the client is hoping for a final product. The supervisors will use this document to grade how we implement and design the project.

D. Design viewpoints

This document has organized into five design viewpoints that will be used to describe the Postal Extension. These viewpoints are as follows:

- Composition Viewpoint
- Logical Viewpoint
- Interaction Viewpoint
- Information Viewpoint
- Interface Viewpoint

E. Design Elements

The chief design elements present in the Postal extension can be summed into the following categories: the IDE/extension functions, the UI entities and functions, the Parser and data handling. Many of the attributes of the IDE and extension are related to functionality provided by VSC or functionality that this extension will provide. The UI is broken down into the File Map and Error List viewer, which provide the user with relevant information on their project. The Parser acquires data from the user's files and transfers that information to the data handler functions. The data handler entities and functions provide the UI with up to date information for the user to view. These design elements will be described within the context of specific design viewpoints.

F. Design Rationale

The Postal Extension project will attempt a pseudo agile design style focusing on incremental releases and feature implementation. As this project has a very short time line, it is important to release and mature key features before lower priority features. It an attempt to avoid over-engineering and feature creep, the project has set relatively simple minimal goals that must all be completed before the project can move on to more complex ones. We hope that by setting this restriction, we can guarantee at minimal feature release and meet our time line goals. The current minimal features are (as of 12/01/2016) already under implementation and are expected to be completed before January of 2017.

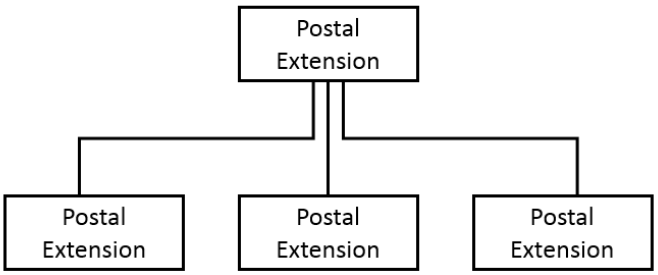


Figure 1. Components of the Postal Extension

G. Design Languages

The following document makes us of Entity-Relationship (ER) diagrams to represent information and data structures and UML diagrams for displaying system components.

V. Introduction of Design Viewpoints

The design components of the Postal Extension will each be addressed and detailed by at least one of the following design viewpoints: Composition Viewpoint, Logical Viewpoint, Interaction Viewpoint, Information Viewpoint and Interface Viewpoint. In each of the following sections, each viewpoint will be briefly described, the design concerns of that viewpoint introduced, then the design elements discussed.

VI. Composition Viewpoint

The Composition viewpoint describes the way the design subject is (recursively) structured into constituent parts and establishes the roles of those parts.

A. Design Concerns

Show the major components of the extension and their general, high-level functions.

B. Design Elements

1) Extension: Type: system Description: The design of the extension is meant to help new web developers make better design decisions when writing HTML and CSS code. The extension is to be built on top of the Visual Studio Code IDE and the extension will be the completion of the all the parts that are described within this document. The parts of the extension are as follows:

- Visual Studio Code and how it interacts with the extension.
- The parsers that take the language and parse the languages and find errors.
- The data structure that is used to store the information that the parser finds.

- The UI that the extension adds to the users.

All of these components make up the entirety of the extension, and they will be described more entirely within this document.

2) Parser: Type: component Description: The parser is the primary logic of the Postal extension. It is responsible for gathering and analyzing all the data from the user and generating the data that will go into the data structure. It will parse through each file and check if the user's code violates any best practices of HTML, CSS, JavaScript, or PHP. It will also generate the file map by looking at what HTML pages are linked to other HTML pages. The parser will update the data structure with only the data that was changed most recently. For example if a user saves a file, only that file be parsed again and its new data sent to the data structure. Similarly if a user saves multiple files at the same time then only those file would be parsed and sent to the data structure.

3) File Map and Error List UI: Type: component

Description: The FileMap and Error List UI is the only GUI included in the Postal Extension. The GUI has two primary responsibilities:

- Displaying the both a visualization of the user's project directory in the form of a graph of interconnected nodes.
- Displaying a list of the Broken Rules in the project directory detected by the extension parser.

Both subcomponents of the GUI will offer a degree of interactivity with the user. The GUI will be launched from the Visual Studio Code IDE through the use of the VS Code Command Line. When Launched, the UI Component will interface with the data handling component to retrieve the information necessary to construct the file map and error list.

4) Data Handling: Type: component Description: The information source from which the UI derives its data will be called the data structure. The data structure is a dictionary of file nodes and links collected by the Parser for the project currently loaded into VSC. The file nodes within the data structure will contain information related to all the files in the projects, as well as error data detected by the parser. The data structure will be serialized into structured JSON objects by the JSON.stringify function and saved to a file. [2] This JSON file will be the direct source from which the UI obtains its data.

VII. Logical Viewpoint

The purpose of the Logical viewpoint is to elaborate existing and designed types and their implementations as classes and interfaces with their structural static relationships. This viewpoint also uses examples of instances of types in outlining design ideas.

A. Design Concerns

Show the abstractions at the class and datatype level that are required for each component.

B. Design Elements

1) Parser: The Parser will be implemented using Perl and the HTMLTokeParserSimple library[3]. This library tokenizes HTML files into their basic parts such as the tags in it as well as the various attributes within each tag. The parser will use this information to grab links from the pages and determine which pages are linked to each other. It will also look at a higher level of the users code structure and determine if they are violating any best practice thus creating Broken Rules.

The Broken Rules will be determined by going through the W3C best practices list and determining what types of things should exist in each file type. For example having only the structure of the website in the HTML pages and having all the styling in the CSS pages. We will then format rules in which our parser will interpret and return error messages to the user if they broke a best practice thus generating a Broken Rule.[4]

2) File Map and Error List UI: The User Interface system will consist of two main components: The File Map and the Error List. Both of these components will be bundled together into a single screen. This screen will implemented in an electron application window. Electron is a platform used to create desktop applications as if they were websites. [5] The user interface will then be implemented using JavaScript, HTML and CSS.

3) FileMap: Type: Interface Description: The file map will be a graphic representation of the of the user's project solution. It will appear as a web or graph of interconnected nodes where the nodes represent a file in the user's project directory and an edge represents some link (defined in the parser section) between the two files. This web will feature nodes of different sizes and will allow the user to zoom and pan the view. The file map will be generated from the above mentioned data structure. On execution of a Visual Studio Code command, the Electron application will fetch the data structure and generate the file map by traversing the 'FileStruct' graph structure. When a Node is generated it will have several visual attributes which will be acquired from the data structure:

- The size of the node will be based on the number of links to that object (size of the links[] array).
- A color corresponding to the type of file. See below table.

Color	File Type
Blue	HTML
Green	CSS
Purple	JavaScript
Yellow	Image
Red	PHP
Grey	Undefined

The name of the node will be retrieved from the file struct name field and will be displayed as text inside of the node. An asterisk will appear next to the name text within the rendered node if there are errors within the FileStruct for that node. In other words, if the size of the errors[] array within the FileStruct is not zero. The file map will be rendered within its own div using the vis.js library 'Network' module. The Library by default includes the rendering, panning and zooming functionality. [6]

4) Error List: Type: Interface Description: The error list will display all errors currently in the project directory in the form of a vertical list. These errors will be retrieved when the UI opens from the same data structure is being generated. These errors will be retrieved in a per node fashion and will also be grouped in the error list in the same order.

The error list will exist to the side of the file map in the same electron application screen. The list will allow the user to scroll when the number of error result in the list exceeding the electron window height. When an error in the list is hovered over, these errors will highlight the corresponding node in the file map by changing the color value of said node. When an error in the list is clicked, the extension will open the file in Visual Studio Code's text editor and scroll to line where the error exists. The error list will be in its own div and scrolling functionality will be achieved through the use of JQuery Advanced News Ticker. [7]

5) Data Handling: Data handling within the Postal Extension consists of three main entities or processes: the data structure, serialization of the data structure and the storage of the data structure in a JSON file. The data structure is a dictionary of file nodes, stored as JavaScript objects. These JavaScript objects come from the Parser parsing the currently loaded project for links and errors. The data structure can be considered the live version of the project data, as its dictionary is updated by the parser every time the project is saved. Once the data structure is updated, its data will be compared with the now out-of-date JSON file to identify which file nodes were changed. The comparison of JSON and JavaScript object will be done with the Lodash library's deep object compare function. The information concerning which nodes were changed will be passed to the UI elements, once it has been serialized to the file. The nodes changed in the data structure will then be serialized into JSON using the JSON.stringify function built into JavaScript. This JSON will be saved to a file, which can further be read by the UI functions for updating. [2] [8]

VIII. Interaction Viewpoints

The interaction viewpoint defines strategies for interaction among entities, regarding why, where, how, and at what level actions occur. Most of these interactions are between predefined events and event listeners.

A. Design Concerns

The primary design concerns from an interaction viewpoint in the Postal Extension are within the IDE and its interactions with the rest of the components of the software. Additionally, within the data handling elements of this extension, how the data passes from the data structure to JSON files is also an important interaction.

B. Design Elements

1) IDE: There are three main interactions that happens on the specific IDE events; the parsing of the files ,the opening of the custom extension UI, and the opening of an error object. The first of these events is the parsing of files. There are several actions that occur within VSCode that will initiate the extension parsing and interpreting process. These specific events that will have specific event listeners. Each event will trigger a specific type of the same parsing process. These events are when the user starts the extension, when the user explicitly saves one or all

the files, and when the user closes the application. When the user starts Postal the extension will first initial parsing and create the data structures that will serve as a reference for the next continuation of the extension processing. The first parsing will be set in motion by the built-in activate function with the extension initialization. After the initialization whenever the user saves the files the extension is going to parse the files and get the necessary information from the new parse. This will continue after every manual save. The extension will only continue on the explicit manual saves, meaning only when the user to saves the files, rather then when VSC auto saves. Once the user saves the files the inPerSaveDocument listener will be acted upon.[9] This specific event will allow the extension parser to read the files contents before the file is actually saved. This will allow for the extension to possibly format the user's code before it is saved. The third even is when the user closes the VSC application or kills the extension then the deactivate listener will do one last parse of the files so that the user will be where they left next time they comeback to the project. These three events should be able cover the major of instances when the extension is expected to be iterated.

The extensions custom UI will open on several events. The first event that will bring up the UI will be the user using the open UI command. The user will be able to use the VSC command to open the UI. Unless the parsing operations has not been invoked on the current working directory this command will just bring up the UI using information that was created from previous parsing. This will ensure give the user access to the UI at any moment, and the UI won't become a bother to the user popping up after every save. The other event that will cause the UI to be brought up will be when the parser identifies a new error in the code. This will make it to the user can quickly identify the errors that they are creating before there become too many that the user is overwhelmed by the error table once they check the UI.

An error object is the location of what the extension finds and identifies as an error. These error will consist of improper HTML and CSS practices. These errors will be listed in the custom UI, and if the user interacts with the error object the extension should navigate the user to the location of the error. The extension will do this with openTextDocument, to first open the document that has the error. Once the extension opens the file it can then get the specific words that are part of the error and change the background color using the background color property. [9] This will hopefully give the user a good idea of where the user has created the error and the errors information text will give the user an idea of what mistake they made and how it can be fixed.

2) Data Handling: Most major actions within the data handling entities and functions are reliant on the Parser being called and providing updated data. The data structure is updated when files are saved and reparsed. Information on which file nodes within the data structure dictionary were changed is also identified when a reparse occurs. The likewise is true with serialization of the data structure to the JSON. Each Parser call results in each data handling function taking place. Other components that rely on information within the data handling scope are then also reliant on the Parser for updates.

IX. Information Viewpoint

The Information viewpoint is applicable when there is a substantial persistent data content expected with the design subject.

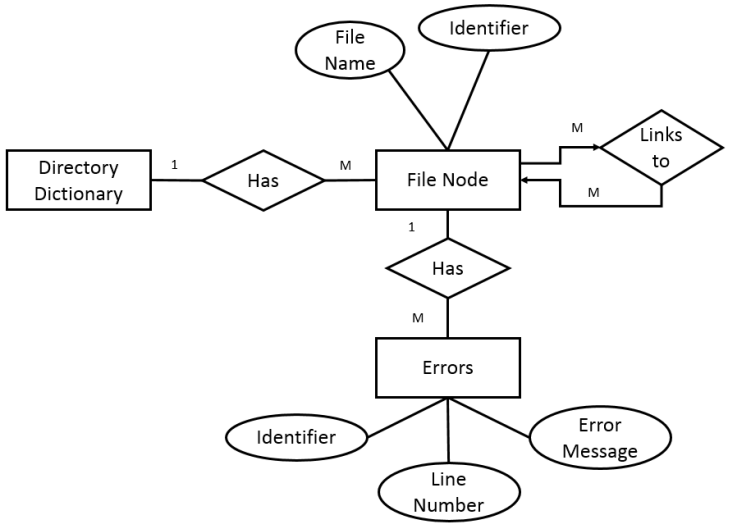


Figure 2. A visual representation of the data structure.

A. Design Concerns

The primary element of concern from an Information Viewport in the Postal Extension is the data structure data layout. The main objects contained in the data structure are detailed below.

B. Design Elements

1) Dictionary: The overall format of the data structure is as a dictionary, that is, a collection of key-value pairs. The keys will be identifiers and the values will be specific file nodes. At this point in time, the exact implementation of the dictionary keys has not been determined, but it will likely be some sort of hash table. The values, the file nodes, will contain the relationship and error data for all files in the loaded project.

2) File Nodes: A file node will be representative of an individual file in the loaded project. Each file node will contain the name of the file it represents, as well as a numerical identifier for lookups. File nodes will be linked to each other according when one file in the project references another. Many files may be linked to many other files. The files supported by default will be HTML, CSS JavaScript, PHP and image files.

3) Errors: File nodes will also contain any and all errors present in their respective files. These errors will have a unique identifier, the error message or type of error occurring, and the line number the error occurs on. File nodes may have multiple errors, but each error is only linked to one file node.

X. Interface Viewpoints

The Interface viewpoint provides information designers, programmers, and testers the means to know how to correctly use the services provided by a design subject. This description includes the details of external and internal interfaces not provided in the SRS. This viewpoint consists of a set of interface specifications for each entity.

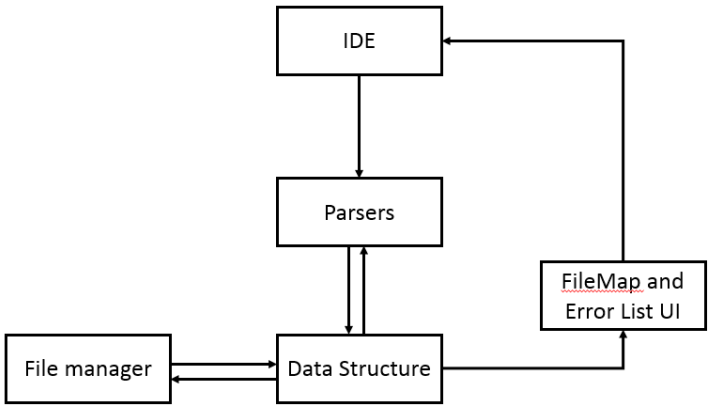


Figure 3. Exposed interfaces

A. Design Concerns

Identifies the interfaces that the components of the framework expose or require to achieve their functionality.

B. Design Elements

C. IDE

1) Exposes:

- Opening the GUI give the user an interface in which they can interact and see the information that they are looking for from the Postal extension.
- That information includes the file structure and how they relate to each other, as well as the possible errors.

2) Requires:

- The parsing of the file is the process that populates the information that is displayed in the UI.
- The operations are signaled to start on certain event listeners.

D. Parser

1) Exposes:

- Parsing for links and Broken Rules

2) Requires:

- The parser will be triggered when the user saves files, and only those files will be parsed. It will also parse on first launch of the extension.
- Get Data from the Data Structure

E. Data Structure

1) Exposes:

- Get Data

  The data structure has its dictionary of file nodes updated when the Parser reparses the loaded project. The data structure gets its data from the parser when Get Data is called.

2) Requires:

- Load Data (Files)

  The data structure will compare its updated data with the data stored in the JSON file. In order for this to occur, the data structure must make use of JSON.parse and the deep object compare as part of the Load Data function. [2]

- Parse (Parsers)

  The data structure will only have its data updated when the Parse function is called.

F. User Interface

1) Exposes:

- Error Navigation When the user clicks on an error item in the error list, The UI component will interface with the VS Code IDE API in order to navigate the user's screen to the file and line number of the error.
- User-FileMap Interaction The User has several options to interface with the File Map. When the user scrolls a mouse wheel, the file map will zoom and expand the size of the file nodes. If the user clicks and drags on the white space in the background of the file map, the GUI view will pan and render/discard file nodes that are off screen. If the user clicks on a file node and drags, the UI will simulate two dimensional physics and will warp the file map in response to the users movements. The three above features can all be achieved through the use of the vis.js API.

2) Requires:

- Get Data (data structure)
- The UI will need to use the command that is used by the user to view the file map and the GUI.

G. Files

1) Exposes:

- Load Data The JSON.parse function will be utilized to pull data from the JSON file. Once the information about the file nodes from the data structure are pulled from the JSON file, it will be compared to the current data structure values to identify which links or errors have changed. [2]
- Save Data Once parsed data from the Parser is transferred to the data structure using the Get Data functionality, this data will be serialized into a JSON file with the JSON.stringify function built into JavaScript.

2) Requires:

- Get Data (data structure) It is necessary for the data structure to obtain data from the Parser each time it parses so that this data can, in turn, be saved to the JSON file.

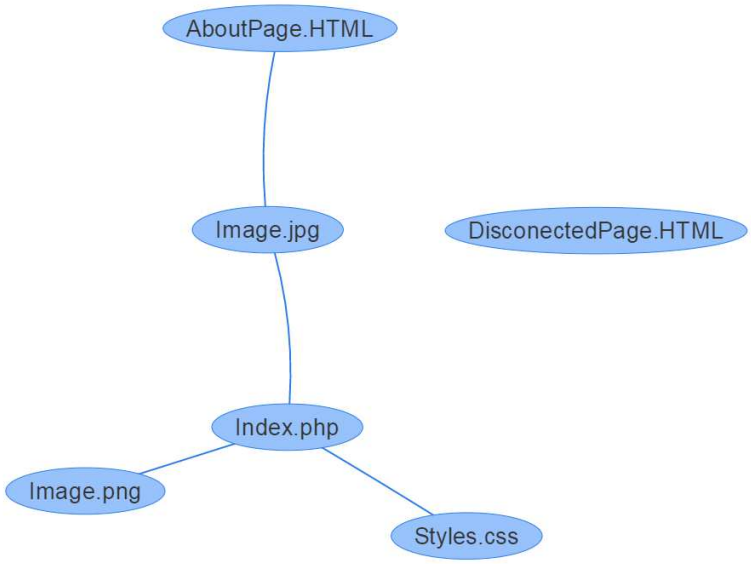


Figure 4. Mockup of the user interface

REFERENCES

[1] Apple. (2016) Model-view-controller. Apple. [Online]. Available: https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaC

[2] Mozilla Developer Network. (2016) JSON.stringify(). Mozilla. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify

[3] C. O. Poe. Html::tokeparser::simple. [Online]. Available: http://search.cpan.org/~ovid/HTML-TokeParser-Simple-3.16/lib/HTML/TokeParser/Simple.pm

[4] Best practices for authoring html current status. W3C. [Online]. Available: https://www.w3.org/standards/techs/htmlbp#w3c_all

[5] (2016) About electron. Electron. [Online]. Available: http://electron.atom.io/docs/tutorial/about/

[6] (2016) Network examples. visjs.org. [Online]. Available: http://visjs.org/network_examples.html

[7] V. Ledrapier. (2016) About electron. [Online]. Available: http://risq.github.io/jquery-advanced-news-ticker/index.html

[8] Lodash. (2016) Lodash Docs. Lodash. [Online]. Available: https://lodash.com/docs/4.17.0

[9] Microsoft. (2016) Visual studio code api reference. Microsoft. [Online]. Available: https://code.visualstudio.com/docs/extensionAPI/vscode-api

Client Signature

Signature

Date

Student Signatures

Signature

Signature

Signature

Signature

Date

A. Discussion of Changes

The original design of the Visual Studio Code extension was targeted towards web developers. Discussions with the client transitioned this viewpoint to make the tool more universal and available to a wider audience of developers. The tool was then redesigned to allow for broader customization through the use of "grammars" which users can use to define certain behaviors for the extension. This change allowed for more customization and tuning available to the user, meaning it didn't really matter what kind of developer they were because they could adapt the tool to their needs.

Another change from the original design was the use of certain technologies which will also be discussed below. The developers had intended to use the programming language Perl and associated libraries to parse the project directories of the user but found keeping the entire project written in TypeScript made more sense and was easier to implement while keeping the same amount of functionality. Other technology changes included using a custom implementation of the features provided by jQuery's Advanced News Ticker API library.

One major design change from the original document was not parsing the updated file on save or having multiple versions of the parsed data structure around. It was decided to only parse the project directory when the user explicitly said to (when the user wanted to refresh the file map UI or notification list).

It was also briefly mentioned in the original design document that the extension would reformat the user's code. This feature was removed.

V. Technology Review and Changes

The original Technology Review Document for this project is inserted in the following pages.

A Tool to Automatically Organize the Structure of a Codebase Using Information Foraging Theory Design Patterns

Technology Review and Implementation Plan

Team Postal — Group #38

Cramer Smith, Sam Lichlyter, Eric Winkler, Zach Schneider

November 16, 2016

CONTENTS

I. Introduction

The following document will compare already available tools we have considered and our decisions on the best tool for our product. Sam will be looking at the HTML Parser and the various tools required to perform those actions. Zach will be looking at various ways to store our data structure as well as its serialization and the comparison between the two. Cramer will be comparing the various text editors we have to build our product around as well as the best way to handle events inside the text editor and what language we should build our tool with to best utilize the text editor's provided functionality. Eric will compare various visualization libraries, as well as the best way to display the rules the user has broken, and how to display the file map.

II. Sam Lichlyter

A. Parser Class

This is the class we will use to actually parse the files in the directory and translate it into the data structure we decide to use.

1) PEG.js: PEG.js is a parser generator in JavaScript. Its sole function is to generate a parser which is saved as a JavaScript Object that can be interacted with. It also generates a small API. This would reduce the work needed to create a parser, which would allow for more time to be dedicated to other areas of the project. The problem with this is that it would only generate a simple API. Theoretically the API it generated could be altered, but this would then require more time dedicated to learning how the generated parser was built and how to successfully build in the required changes. [1]

2) GOLD: GOLD is also a parser generator. It however does not generate to JavaScript. GOLD takes in a grammar, which we would come up with that reflected the various web technologies we would support and then it outputs an engine in which we would use to parse the files we were given and output our data structure. This would require us to figure out how to use this engine once we get it generated and test it against the various scenarios we expect to come against. It could however catch various corner cases we had not expected, but this is unlikely since we would have to come up with the grammars. [2]

3) Custom Parser: The pros of a custom parser are that we would only include what we needed to. This would reduce a massive overhead in interacting with any API we would have to use with a third-party tool. It would also mean we could optimize the algorithms we use for it specifically so it could be faster than the third-party tools. This would also reduce the number of dependencies our extension would rely on greatly meaning there is fundamentally less to go wrong should one of the dependencies change. For these reasons I think this is the route we will take.

B. Parsing Language

This is the language we will choose to use for the parser. This was fairly dependent on the parser class we chose to use, but since we are planning on writing our own, we have a little bit more flexibility to choose which language would be best.

1) Perl: Perl from the beginning isn't that great of a choice because not very many of our team members have experience with it. However it does do really well as a parsing language. There are multiple examples of excellent parsers coming out of Perl. For example Markdown[3] was originally written in Perl. Perl was written for parsing in mind. There are many articles explaining why regular expressions will not be sufficient enough to parse HTML.[4] For these reasons, at least part of our parser will be written in Perl.

2) Python: Python includes an HTML parser built-in, but it is not nearly as robust or as extensive as the ones written for Perl. The Python library only looks for starting and ending tags. It has a small amount of error reporting, but for our tool, these will not be sufficient. [5]

3) JavaScript: Writing a parser in JavaScript would be ideal since most of our tool will also be written in JavaScript. In the end the parser may be required to be JavaScript because of the restrictions placed on us by VSCode. This is fine because there are many tools included in Node.js and npm that also parse HTML similar to some of the Perl tools. [6][7]

C. Perl Parsing Tool

This is the Perl tool we will be using to parse the HTML which our users want analyzed. This tool must be able to parse complicated HTML in a reasonable manner and give some form of error feedback.

1) HTML-Parser: The HTML-Parser is the canonical Perl HTML parser. It will recognize the various forms of markup on an HTML page as well as separating it from plain text. This tool also parses HTML similar to how web browsers, which is not always how the W3C specifies it should be done. The HTML-Parser will also allow us to turn this feature off, which will allow us to check against the standards of the W3C which is what our tool is mostly about. [8]

2) HTML-TokeParser-Simple: The TokeParser parses HTML and tokenizes the HTML page. This is another popular way to parse HTML pages, but for our intents, it may not be the best solution. We don't care much about tokenizing the input as much as verifying the input is standards compliant as well as throwing it into our data structure. Another problem with this tool is it says the tokens are not always intuitive to parse, meaning there could come extra overhead to get this to work. [9]

3) Parse-RecDescent: The RecDescent parser isn't really a parser, it's a parser generator similar to GOLD mentioned above. It has most of the same pros and cons that GOLD did. We would have to pass it a grammar, which again probably wouldn't catch many corner cases because we would be creating the grammar. [10]

4) Decision: The HTML-Parser is our best stand-alone option. However, we are not limited to just one of these options. Using the HTML-Parser to parse our HTML and check if it is standards compliant would be the best use case. We could also use the TokeParser to tokenize our HTML input which might be extremely useful for building our data structure.

III. Zach Schneider

A. Data Structure Storage Medium

The text editor extension created for this project functions in part by parsing an existing code base. The parsed code base will be converted into a sort of data structure that can be more easily utilized and manipulated for helping

the user, while not interfering with their actual files. The user's code base will be parsed periodically, but that parse function may be resource intensive, and as such, serialization of the parsed data into a more accessible form will be required. The following sections will evaluate SQL vs. NoSQL data storage options, as well as the method for converting the parsed data into a storable format.

1) SQL Databases: SQL Databases have historically been the go-to method for data storage for many computing applications. SQL has been around for many years, resulting in great familiarity with it and its resources among developers and our team. SQL databases are often based on tables and predefined schemas, contrasted with NoSQL databases which often lack a predefined structure and consist of key-value pairs. The primary issue with a SQL database in our project is the overhead required in installing and maintaining a local database. Databases may be hundreds of megabytes just to install, and setting up a database connection may not be natively supported in or application. For these reasons, our extension will opt not to use a SQL database. [11]

2) NoSQL - JSON: In contract to a more structured SQL database, NoSQL databases may provide the flexibility, native support and light weight file structure our team needs to efficiently store parsed user data. NoSQL databases often store data in the JSON format, a human and machine readable file that is ideal for temporary data storage and exchange. JSON is built into JavaScript, one of the languages that will be used in our extension, meaning no additional setup will be required to send parser data to this storage format. Additionally, JSON is natively supported by most modern text editors and browsers, including the one being considered for this project. JSON can store complex JavaScript objects and variables, making it the ideal choice for containing the data structure utilized by our extension. [12]

3) NoSQL - BSON: BSON is a binary-encoded JSON data format utilized by MongoDB, a NoSQL database. BSON provides additional data types to the database for "efficient encoding and decoding within different languages." [12] MongoDB is able to manipulate BSON files to create additional structures inside the file, even after object hierarchies have been created. MongoDB is a free and open-source database solution that uses BSON documents for storing data. It is commonly used in big-data or real-time applications, often used in conjunction with node.js and other JavaScript technologies our development team is familiar with. Two main problems exist with BSON: it is not human readable, making it harder to debug, and it requires a full installation of MongoDB, which brings the same space and overhead concerns mentioned before. These additional challenges have convinced the team to stick with a purely JSON solution. [13]

B. Data Structure Storage Serialization

1) JSON.stringify: Since the text editor will be written in JavaScript or a superset of it, using JSON as the method of storing objects seems the most obvious. JSON text is valid JavaScript code and is supported in many modern languages and IDEs. JSON is also plaintext, which requires significantly less overhead and space than a full database. As of the ECMAScript 5.1 standard (2011), the JSON object in JavaScript has a built in stringify and parse function, which will serialize and deserialize JavaScript variables and object respectively. As this function has been officially supported for many years and has multiple sources of documentation and example usage online, it will be the primary choice for how the extension stores files related to user projects. [14]

2) serialize-javascript: A shortcoming of JSON.stringify is that it does not allow actual JavaScript functions to be serialized into JSON, nor does it allow regular expression statements. The serialize-javascript npm package serves as a superset of JSON.stringify while also including the aforementioned functionality. Additionally, serialize-javascript will auto escape HTML code, making the JSON safe to display on webpages in raw form. Npm is supported by dozens of IDEs and text editors, so compatibility will not likely be a problem. The largest drawback of using serialize-javascript is its lack of documentation, despite its mild popularity. There are fewer examples of its usage online than JSON.stringify, and it adds another external dependency to our extension. For these reasons, serialize-javascript will not be a part of the storage system in the extension. [15]

3) JSON-js: JSON-js, also known as JSON in JavaScript is the former implementation of JSON support for JavaScript in web browsers. It provides conversion of JavaScript data to the JSON format, as well as the reverse parsing functionality. It is backwards compatible for browsers all the way back to Internet Explorer 8 with its json2.js file. However, these files have been superseded by native JavaScript support of JSON since ECMAScript 5 was made the web development standard in 2009. Even Douglas Crockford, the author of this utility now recommends against its use. For this reason, our team will opt for the natively supported JSON.stringify function. [16]

C. Comparing the Data Structure to the Serialized JSON

Once the parsed user code base has been serialized into a JSON file, the extension will then need to periodically check if the data objects in memory are different than the objects saved to the disk in the JSON file. There are dozens of methods to compare objects in JavaScript, some prioritizing speed with others focusing on functionality. Our team opted for somewhat of a middle ground, while leaning towards functionality and ease of development when comparing the Lodash library, the JSON.stringify comparison function, and the deepEqual npm module.

1) Lodash: Lodash is a JavaScript library the provides a wide range of functionality while focusing on performance. Lodash abstracts iteration through arrays and objects to maintain speed while simplifying the experience for the developer. The library contains functions such as .cloneDeep() and .isEqual() which allow for deep object comparison (instead of the native ===), as is needed for this extension's circumstances. Lodash has continuing developer support and sufficient documentation throughout web. The main detraction for using Lodash is that it creates a new dependency on an entire library, something our developers would have liked to avoid. However, the benefits of Lodash seem to have outweighed that primary flaw enough for this library to be our choice method of data structure comparison. [17]

2) JSON.stringify: JSON.stringify, as previously mentioned, is native to JavaScript and will be our primary method of serialization to JSON files. The usage JSON.stringify(a) === JSON.stringify(b) can compare more deeply than === alone, caring about the contents of the objects it serializes rather than the structure or referential equality. JSON.stringify can also compare more efficiently than the deepEqual package according to this article. [18] The main downside in relying on JSON.stringify to compare our data structure to existing JSON files is that the rest of the legwork in finding what was different would still be up to our team to develop. Lodash, instead provides much of this functionality, leading to it being chosen over JSON.stringify. [14]

3) DeepEqual: The deepEqual deep object compare function is a free library available via npm. It is quite popular among JavaScript developers and has a decent amount of documentation online. Like JSON.stringify, it does a deep comparison on two JavaScript objects checking for differences. However, it is not only slower than JSON.stringify, but it also has the same lack of functionality that JSON.stringfy suffers from. For these reasons, our team opted not to use deepEqual. [19]

IV. Cramer Smith

A. The Best Base Integrate Development Environment

It is necessary to look all the possible integrated development environments for the postal project to see what would be the best fit for this specific extension, both development wise and release wise. To this purpose this review will examine three similiar IDEs that could all be used as the base of the extension that is planned to be implemented. Those IDEs are Brackets, Atom, and Visual Studio Code (VSCode) all of which have there advantages and disadvantages.

1) Brackets: Brackets is a text editor owned by Adobe, and is currently in development as an open source project. [20] Brackets is made specifically for web development, and offers tools such as the Chrome debugger, inline editor, and live website previewer built in. The research into the Brackets IDE has brought to attention many features that were good, but also some that were not as good. Adobe started the development in 2011 making Brackets the oldest of the IDEs research in this paper meaning it is the most established and it has had more time to become a more stable build. The longer life of Brackets could also explain the more extensive documentation that it has when compared to other IDEs extension development documentation. Another nice feature of Brackets is that it has dedicated API functionality that allows for extensions to safely modify the underlying Document Object Map (DOM) which would be very useful for the Postal extension. While there are several benefits to Brackets there are some drawbacks specifically being the extension debugging and lack of usability in the extension manager. The extension debugging consists of having another development environment open with the extensions code and restarting Brackets with every change. This restarting will get tedious after prolonged development. The other issue is the extension manager in brackets is not user friendly, it is basically a list of extensions and a search bar. With the target audience Postal is trying to reach is one of rather new web developers, people who do not want to sift through extensive lists of confusing extensions. Brackets would make it more difficult for our product to get to the users.

2) Atom: Atom very similar to Brackets in that it is a text editor developed by GitHub that is currently in development as an open source project. It features cross platform editing, a built in package manager, and smart auto completion. Atom advertises itself as the 'hackable' text editor meaning that it is made using HTML, CSS and JavaScript in such a way that anyone is using the text editor for web development then they should be able to develop for Atom. [21] This is a commonality between all of the possible IDEs. Atom is actually what VSCode from, but VSCode added TypeScript to the languages that it is built in. While atom is a good has a editor and auto completion it does not stand out when compared to the other IDEs. In fact it seems as though Atom uses extensions as a crutch not implementing built in functionality requiring the user to get extensions in order to complete their

tasks. While this does make the initial learning curve of using Atom a bit larger it does keep the editor light and running very fast. What Atom gains in speed it loses in ease of use as a new user has to do a good amount of initial set up and for the audience that Postal is targetting would be put off by the confusing intial set up of Atom.

3) Visual Studio Code: Visual Studio Code, like the other IDEs, is a text editor by Microsoft, that is currently in development as an open source project. [22] It is the editor that was initially proposed as the base of the Postal extension. This initial idea to using VSCode was a product of the team having worked with Visual Studio and enjoying the experience. Now that the team has tried working with Visual Studio Code there has been some benefits and drawbacks identified. The benefits were that VSCode has justifiably better extension debugging than the other IDEs available. VSCode seems to build around the idea that people will be making extensions for Visual Studio Code so there is a development window that is created when debugging extension code within VSCode. The other IDEs seem to be less approachable with a system that makes the developer reinitialize the IDE every time the extension's code is changed. The other benefit of using VSCode was that the extension can be written using TypeScript, but that being said none of the team members have used TypeScript meaning it would be additional learning curve added on to the obstacle of learning more JavaScript. The first of the drawbacks of VSCode become evident when working with the documentation and finding that Visual Code is fairly new, and does not have a lot of documentation or examples of extensions to readily examine. As this team is fairly new to creating extensions and not efficient at writing in JavaScript or TypeScript this is going to be a problem. Visual Studio Code also had no built in API for modifying underlying DOM that make up the main user interface which the postal extension possibly could do.

4) Decision: All this being said the team has decided to use Visual Studio Code based on several major benefits. Visual Studio Code's use of TypeScript and being able to import .NET libraries. All of the team members have at least some experience with .NET libraries, and these could be used to greatly improve the project. The Visual Studio Code also has the best Extension Debugger than the other IDEs and a more defined extension creation process. This is important for the integrity of the development of the extension in the long run. Visual Studio Code has a extension creation guides the developer through the process of making the extension and creates all the helper files that the developer will need in their extension. The other benefit is that the built in extension manager will be easy for the user to manage than the other IDEs.

B. Event Handling Within A Separate Window of the IDE

Visual Studio Code is going to interface with the user and our extension will need to know what the user is doing. The specific instance that this portion focuses on the the event handler that the IDE is actively listening for from it's extension. The Postal extension needs a way of getting information from the extension, this section will explore the options that the VSCode extension API offers developers for this kind of interaction. The events that the IDE will be listening

1) HTML Preview Links: The first way that VSCode makes it possible to have events is through the HTML Previewer. [23] This method would require that the extension makes an HTML page and then display that page either in a web browser or using the built in command previewHtml using the vscode.executeCommand() and

passing it an Uniform Resource Identifier (URI). The URI that would be past to the command would have a HTML DOM that would be rendered in a separate panel on top of the VSCode's HTML and CSS. This seems like an easy way of creating a user interphase, but the only intractable element would be links. Links would only be able to change the the view to other HTML files. The idea of making every possible use case a separate HTML file would be difficult, and extremely tedious.

2) IDEs Built In Events: Visual Studio Code has a number of already built in events that the extension could listen for but these events fire at specific timing such as when the user saves a file or edits a document. The Extension could fire these events that are built in to Visual Studio Code, and have handlers set up to listen to those events using the already built in listeners. This method would be extremely problematic. These built in events are meant to be used at very specific times and have set listeners that parts of the IDE itself, and other extensions are expecting at very specific times. Using the events would be an incorrect solution as these events happen at every specific instances and using these events for any purpose other than the purpose that they are currently made for would be irresponsible and would not work properly.

3) EventEmitter: The better option would be to use the VSCodes EventEmitter to create and manage for other to subscribe to. With an eventEmitter the extension can create listeners that can then be listen for when the event fires signifying to the listeners that the event has occurred. This is the intended way that exertions are supposed to create and the event handlers. There are also built in events that the these event listeners will need to be initialized before the user actually interfaces with the extension and to do that the extension will need to use the activation events. An activation event is set in the package.json of the extension and these activation events are sent from the IDE to the extensions. These events can be onLanguage, onCommand, onDebug, workspaceContains or a star (*) signifying that the extension should always be running. When the extension recieves one of these events that it is listening for it will start the extension operations. Setting these custom eventEmitters and event listeners should be the first thing that the extension sets up as they will be very important for the interface that the user will interacet with. [23]

4) Decision: For the purpose of this project the eventEmmiter and Listener will be used for obvious reasons. The first being that it is the standard of Visual Studio extensions, and this is what extensiona are expected to implement. The other proposed option could work but would be improper and irresponsible. The HTML preview method would be extremely limited, tedious and impractical, and the built in events would go against the design of the IDE and would cause conflicts that would be extremely problematic. Make custom eventEmitters will allow for greater flexibility and better design in general.

C. Languages Extensions Written in

All the possible IDEs that could be used for this project are written in HTML, CSS, and JavaScript, but the bulk of the logic is JavaScript. This means that the extensions use mainly Javascript. With Visual Studio Code there is the option to use either JavaScript or TypeScript. This portion will look at JavaScript and TypeScript looking at each language's pros and cons in terms of the development of the Postal extension.

1) JavaScript: JavaSctipt was created in 10 days in 1995.[24] It is a well known, and well documented language. With all the possible IDEs the extension can be written in JavaScript. It can be very confusing at times as it is an asynchronous language and not compiled inline. This compounded with the fact that all the extension documentation for VSCode is written with TypeScript in mind making the learning curve for making an extension in JavaScript that much steeper. The upside would be that there are in more people programming with JavaScript and that it has a larger community to gather information from when compared to TypeScript. This would allow the Postal team more f

2) TypeScript: TypeScript is a language created by Microsoft with the purpose of being 'JavaScript that scales.' [25] In fact TypeScript compiles to JavaScript and its syntax is almost identical to JavaScript. The main advantage of using TypeScript is that it adds types to JavaScript allowing for static checking and code refactoring when developing JavaScript applications. [25] Another major benefit of using TypeScript is the ability to import .NET libraries. This allows for developers to use more powerful APIs then the some of the less robust JavaScript APIs. For this project that will allows the the extension access to APIs that the team has experience using. Another major advantage of using TypeScript is that all the VSCode examples are already written in type script making understanding the documentation easier to understand. TypeScript also allows for the use of JavaScript within TypeScript files and that flexibility will be nice to have while developing an extension.

3) Decision: For the postal extension the developement will be done with TypeScript as its main programming language. The main reason for this decision is that the documentation for VSCode extension developement is written with TypeScript examples making for a smoother developement experience. The other reasons for chosing TypeScript over JavaScript is that TypeScript will allow for the use of both JavaScript and TypeScript. Have the flexability to chose either depending on the circumstance will make for faster developement with more possible solutions to some problems that the developers may face.

V. Eric Winkler

A. Rendering the File Map

The file map is the visual representation of the user's current project. It will most closely resemble a web of nodes where the nodes are individual files and the edges are confirmation of some sort of link or dependency between two files. Whatever technology we use for this must be capable of running the VS Code environment (JavaScript).

1) vis.js: Vis.js is a "dynamic, browser based visualization library". It features several modules for visualizing data in a JavaScript application. One of these modules, the Network module could serve to visualize the file map in a web like format. The Network modules features the ability to click and drag the map around which will be useful for navigation. Even better is the modules ability to zoom further into the map and dynamically change text size of the nodes depending on the zoom level. This is an extremely beneficial feature as it allows the system to draw many smaller nodes without having to deal with the concern of the user being able to clearly read each node at once. Additionally, the module appears to be highly customizable in terms of aesthetic.

Vis.js claims to “run fine on Chrome, Opera, Safari and IE9+.” However as the project is being developed with the intent to within visual studio code, Its ability to run in that environment is questionable. This is currently the primary concern with this option.

Vis.js is open source and is dual licensed under both Apache 2.0 and MIT.

2) d3js: Similar to vis.js, d3js is another javascript library designed to visualize arbitrary data. Unlike vis.js, it allows the user to bind processed data directly to a Document Object Model (DOM). This essentially means that it is more flexible in its ability to read in data. It also features an output to an SVG file type, which could be useful as we know Visual Studio Code is capable of displaying SVGs within the environment.

D3js features a large massive number of what vis.js referred to as modules. There are several suitable modules that D3js offers, but the most promising appears to be either the force-Directed Graphs or the Curved link graphs. Like vis.js, D3.js features zooming and panning. The API reference appears to be a bit less straight forward than vis.js and the library descriptions claim to prioritize making the library light weight and efficient. This option has the potential benefit of being more efficient, but will likely be slightly more difficult to integrate.

D3.js supports “modern browsers” (Firefox, Chrome, Safari, Opera, IE9+, etc.) and also runs on node. The library is available under the BSD License.

3) Custom Code: Our final option is to write custom own code for generating the file map. This option is massively more work than the first two but still has some benefits. Writing custom code allows us more flexibility in how the code runs and allows us to keep the size of the extension to a minimum. Every feature implemented in the custom code will be out of necessity. Additionally, it adds the benefit of not having to worry about properly citing the code according to the licenses above. This is a relatively minor benefit, but also means that copy write will never be an issue for this corner of the project.

B. Displaying Broken Rules

In addition to displaying the file map, the UI will also feature a list of broken rules to one side. These broken rules are pseudo-error our parse detects within the project directory and should be displayed alongside the map. The displays should be as aesthetically pleasing as possible and will ideally provide an API a hook for triggering an event when the user clicks on an object in the display. The errors will be stored as strings, so there should be relatively little complexity in passing the data to an API.

1) JQuery Advanced News Ticker: The JQuery Advanced News Ticker is a JavaScript based JQuery plugin. It claims to provide several callbacks and methods to allow maximum flexibility and implementation. It is called through CSS classes and does feature an API hook for triggering a JavaScript event on clicking an object. It being a JQuery plug in should allow it to work within Visual Studio Code. It features a relatively through API guide and install instructions.

The example news tickers it displays an aesthetically pleasing and varied, indicating that customizing the displays is possible. It features a standard vertical list which will suit this projects needs but also an interesting call out version. The version has on expanded item in the list while the rest are collapsed, saving a decent amount of screen space. This may be something to look more into if the project uses this API.

JQuery Advanced news ticker is available under the GNU Public License.

2) nanoScroller.js: NanoScroller.js is similar to the JQuery Advanced News Ticker in that it too is a JQuery plugin. Its goal is to offer a very simple and uncluttered way to create scrollable divs that are visually pleasant. As such it features far less in terms of animation and visual 'coolness' but is far smaller in size. It is likely a much more efficient option in comparison to JQuery Advanced News Ticker as well. It will accomplish the minimum in terms of displaying errors within the UI. A downside of the plugin is that it doesn't feature explicit hooks for triggering a JavaScript event when something is clicked. This is to be expected as it is basically just a div class but it will still be something we will have to implement ourselves if we go with this option.

Documentation is fairly thorough and the website offers a website compression file which will help keep the extension size to a minimum.

NanoScroller.js is available under the MIT license.

3) Custom Code: The final option is to create our own div class. As discussed above in the file map technologies section, this will be quite a bit more works but has quite a few benefits. Again, the project team will have more control over design, efficiency and features than if an API was used. Additionally, code created would belong to the project and the need to cite could would not be a concern.

C. Technologies for displaying the file map

The file map will be created in JavaScript but there are multiple options for choosing where to actually display it in a desktop environment. Any technologies used in this process should be capable of handling a HTML or JavaScript UI application.

1) Visual Studio Code HTML Preview: In the initial plan for the project, the file map was to render within the Visual Studio Code environment. This option is still valid and the team has already demonstrated from a code perspective in the last prototype that this can be accomplished. The actual process for doing this is slightly unusual however. The code to do this is HTML Preview which is a built in functionality to vs code and is typically used to preview a website. By dynamically creating an HTML page, we can simulate a GUI within VS Code. A downside to this is that the team has already noticed some slightly unusual behavior which may need to be tweaked. Additionally, the size of the window that is used to render the HTML is slightly difficult to control. There is also a major advantage of using the HTML Preview however. The first is that, by using built in functionality, the extension can remain more compact and not have to support itself.

2) In Browser Window: Through the use of some Visual Studio Code API Calls, we can have the IDE launch the user's default browser and load our rendered file map into it. The advantage of this methodology is that the UI can be constructed entirely as a webpage. This provides us with many simple to implement options for creating an interactive UI. The down side however is that the web application UI will have to support multiple browsers and additionally the use of an external program for the extension will make usability inconvenient.

3) Electron: Electron is a framework for creating desktop applications through the use of HTML, CSS and JavaScript. It is kind of like a lightweight browser that doesn't browse but instead servers as a container for applications made with web based technologies. Interestingly enough, Electron was used to create visual studio

code. Using Electron would allow us some freedoms for the UI that might otherwise be restricted by Visual Studio Code's preview HTML Option. As mentioned above, testing the preview option led to some slightly bizarre behavior. Electron could potentially fix those issues. The primary disadvantage would again be that the extension has to include the entirety of the electron framework, making it significantly larger.

REFERENCES

[1] David Majda. Peg.js. [Online]. Available: http://pegjs.org

[2] Devin Cook. Gold parser. [Online]. Available: http://goldparser.org

[3] John Gruber. Markdown. [Online]. Available: https://daringfireball.net/projects/markdown/

[4] A. S. Unur. (2014, Feb.) Parsing html with perl. [Online]. Available: http://radar.oreilly.com/2014/02/parsing-html-with-perl-2.html

[5] Htmlparser - simple html and xhtml parser. The Python Software Foundation. [Online]. Available: https://docs.python.org/2/library/htmlparser.html

[6] tmpvar. jsdom. [Online]. Available: https://www.npmjs.com/package/jsdom

[7] tautologistics. htmlparser. [Online]. Available: https://www.npmjs.com/package/htmlparser

[8] G. Aas. Html::parser. [Online]. Available: https://metacpan.org/pod/HTML::Parser

[9] C. O. Poe. Html::tokeparser::simple. [Online]. Available: http://search.cpan.org/~ovid/HTML-TokeParser-Simple-3.16/lib/HTML/TokeParser/Simple.pm

[10] J. T. Braun. Parse-recdescent. [Online]. Available: https://metacpan.org/release/Parse-RecDescent

[11] Luke P. Issac. (2014) SQL vs NoSQL Database Differences Explained. The Geek Stuff. [Online]. Available: http://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/?utm_source=tuicool

[12] MongoDB. (2016) JSON and BSON. MongoDB. [Online]. Available: https://www.mongodb.com/json-and-bson

[13] ——. (2016) Mongodb faq. mongoDB. [Online]. Available: https://www.mongodb.com/faq

[14] Mozilla Developer Network. (2016) JSON.stringify(). Mozilla. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify

[15] Eric Ferraiuolo. (2016) serialize-javascript. [Online]. Available: https://www.npmjs.com/package/serialize-javascript

[16] Douglas Crockford. (2015) JSON in JavaScript. [Online]. Available: https://github.com/douglascrockford/JSON-js

[17] Lodash. (2016) Lodash Docs. Lodash. [Online]. Available: https://lodash.com/docs/4.17.0

[18] Matt Zeunert. (2016, Jan.) Javascript deep object comparison - json.stringify vs deepequal. [Online]. Available: http://www.mattzeunert.com/2016/01/28/javascript-deep-equal.html

[19] substack. (2015) deep-equal. [Online]. Available: https://www.npmjs.com/package/deep-equal

[20] Adobe. (2016) A modern, open source text editor that understands web design. Adobe. [Online]. Available: http://brackets.io

[21] GitHub. (2016) Atom: A hackable text editor. GitHub. [Online]. Available: Atom.io

[22] Microsoft. (2016) Code editing. redefined. Microsoft. [Online]. Available: https://code.visualstudio.com/Docs/

[23] V. S. C. A. Reference. (2016) Microsoft. [Online]. Available: https://code.visualstudio.com/docs/extensionAPI/vscode-api

[24] P. Krill. (2008) Javascript creator ponders past, future. InfoWorld.com. [Online]. Available: http://www.infoworld.com/article/2653798/application-development/javascript-creator-ponders-past--future.html

[25] Microsoft. (2016) Typescript - javascript that scales. Microsoft. [Online]. Available: http://www.typescriptlang.org

A. Discussion of Changes

Sam discussed the different parser technologies the project would use to parse the files in the user's project directory. He started out with the parser class. It was decided the developers would use a custom parser for the reasons he mentioned in the original Technology Review.

The third section Sam talked about was specifically which Perl parsing tool the team would use. At this point in designing and developing the extension the team was pretty convinced they were going to use Perl as their parsing language. The design change mentioned above where they transitioned the target audience from web developers to a wider audience of developers moved them away from using Perl. Perl had a lot of libraries and built in functions specifically to parse HTML which the extension still needed to do, but it was decided it was better to design a more universal parser.

Zach then went on to talk about how the extension would store its data. He mainly talked about different types of databases including SQL and NoSQL databases. It was decided the team wouldn't use a database at all but rather implement the data structure using a single JSON file which could then be passed to the UI to read to generate the appropriate visualization.

As mentioned above in the design changes, the developers decided to not keep two or more versions of the parsed data, so there was no need to have a technology that compared them.

Cramer looked at different Integrated Development Environments (IDE) and their respective event listeners and languages they are written in to allow the developers a better experience building the tool for that specific IDE. The developers decided to use Visual Studio Code for the IDE they would build the extension for. They did this for the reasons mentioned in the original Technology Review.

The team decided to transition to built in events instead of using EventEmitter for the event listeners. Because the extension is only launched and the directory parsed when the user explicitly tells it to, there was no need for more complicated event listeners than the ones already built in.

Eric discussed the different ways to build and show the UI. The team decided to use vis.js as the main JavaScript engine for displaying the nodes and edges within the file map UI. The broken rules (now called notifications) was planned on being displayed using jQuery's Advanced News Ticker API. This was overruled by custom code as the News Ticker API didn't add much value in terms of usefulness of the extension. The actual location of the file map UI was decided to be done within an Electron window.

VI. Weekly Blog Posts

A. Fall Term

1) Week 3: Cramer Smith: Week 3 Senior Project Work

This week I worked on the problem statement, and thought about the design of the tool that we are planning to make.

We talked to our client about what he wanted and what he was looking for in our plan. We had some initial confusion about what we were supposed to do. Our client Christopher Scaffidi wants us to use IFT design patterns within a the tool. We were more focused on just making the tool itself. After talking to him we are on the same page. We will be using the tool and make a prototype that we might be able to test by the end of the year.

Personally I was part of the conversation that about making the design, we are planning on having a code "tagging" system that will allow users to tag their code for the organization mechanism. This is not the final solution but it is the idea we have

at the moment, and Chris thinks it's good. I also worked on the proposed solution part of our problem statement. That section still needs some polish, but I was assuming this is a rough draft. I tried to define what types of IFT design patterns we would be using within the tool we are creating and define some parts of IFT. I don't think that I was looking from the 10,000 foot level. I need to work on that in the future.

Thus far the tricky part of our project is that the 'problem' is just to use IFT. That is what our client wants. What we have proposed is not a problem but a solution. We want to implement this tool to prove that IFT within an application makes work easier, or at least test that it does. So that's what we are doing now.

For week 4

We hope to maybe start working on our requirements, and resumes. I really want to get started on the coding of the project but I think we do that mostly in the winter? Would it be bad to start sooner?

2) Week 3: Sam Lichlyter: Week 3

This week I set up our GitHub page (welcome), I also worked on our problem statement along with the rest of the team. I have been our main contact with our client, Chris, since he and I have worked together for the past year on similar research projects. This week we sat down with Chris to discuss some of the specifics of the project and to form a rough outline of what he wants to see and when.

Earlier in the week we hammered out a few of the main design decisions for our tool and how we expect the user to interact with it, along with what tools we will be using to build ours, and what languages we want to support at first launch.

Next week we plan on getting our requirements document in a near finished state.

3) Week 3: Zach Schneider: What We Did This Week

This week was one of logistics and clarification. Sam set up our GitHub repo called Project Postal (a weird name, I know) and each of us made a few initial commits to it. I set up the Latex document for our problem statement, so we will have a .tex template file for future documents. We scheduled the meeting time with our TA, Vee, for Fridays at noon. We won't meet with him this week but will in week 4. We have also be in email contact with Prof. Scaffidi, our project sponsor. He has been clarifying some information for us as we wrote our problem statement. We had some initial confusion as to whether the research aspect of our project was the focus, or if the tool we would be developed would simply use principals of the IFT research. Turns out it's a little bit of both, but I think we all have a clearer understanding at the end of this week. Finally, this Friday we are having our first in-person meeting with Prof. Scaffidi to get the problem statement signed and some expectations established for this project. I may update this post if there is information of note from that meeting.

Plans For Next Week

For the class, we were asked to prepare our resumes to bring in, so I will be updated my existing resume this weekend. Besides that, we have no immediate instructions specifically related to our project. As such, we will do as instructed in class Tuesday.

4) Week 3: Eric Winkler: This Week:

1. The group wrote out the problem statement and spent a little time reworking the abstract.

2. Began discussing basic design of how Postal will work and made sure that the project was going to be feasible.

3. Just today we sat down with Chris and discussed expectations of the project. It surprised me that we might actually get a chance to run a study with actual people. REUs FTW.

4. Scheduled the next four weeks. Mostly list what we wanted to have accomplished when and reserved rooms. We're not at the point yet where we're ready to divide up jobs so we'll be working together for the next two weeks at least.

Next Week:

By the end of the next week we'd like to have our requirements nailed down and start to draft out our design for the system.

5) Week 4: Cramer Smith: Week 4

What I did

This week I worked with the team to help define the the problem statement more and start to think about some more basic requirements. I also started looking more at the Visual Studio Code that we are planning to use as the basis of our extension. I was able to easily make a 'hello world' of the extension. And I made my own branch and logged the meeting with a python script that I made it's fun it is working nicely. I need to add some markdown implementation, but I am feeling lazy at the moment. I put it in my HelloWorld Branch.

We met on Wednesday, and worked for three hours hashing out different requirements and the problem statement.

Our Plan For Next Week

Next week we plan to work on the revision of the problem statement and get that finalized, and go to the career fair. We also plan to go work on the requirements more and I will probably work on expanding the hello world extension.

6) Week 4: Sam Lichlyter: This week we revised our problem statement. In doing so we focused more on web developers and refined the goals of our product to reflect that. We decided to create a more visual component which we could use to convey the structure of the users code in a more understandable format.

We also met with our TA, Vee, who gave us some advice on how we should approach the class and our teammates. Being pretty close friends I think we'll be okay and we've had excellent communication in the past, so I'm not too worried about this.

We started thinking about our requirements and the requirements document which we plan on working this weekend.

7) Week 4: Zach Schneider: What We Did This Week

This week our whole team met to discuss our problem statement revisions and to begin to think about what our requirements might look like. During the course of this meeting, we came to the conclusion that we would be better successful if we pivoted our tool towards two more specific categories: web developers and/or inexperienced developers. Previously we had aimed to tackle large scale projects with experienced developers, but we believe this would have been a problem for us time-wise and knowledge-wise. Additionally, it will be easier to test our tool with inexperienced developers (students) than more seasoned developers. We sent an email off to our client to notify him of our pivot (the details still meet all of his original requirements) and we hope and expect that he will approve them.

The team also meet with our TA, Vee, for the first time this week. We all became acquainted and he informed us of what he expects from us. The meeting went well and he believed our project and team were all currently in a good spot.

Plans For Next Week

Next week we will finalize our problem statement and have it signed. The revisions will revolve around rewriting some of the specific terms to be in accordance with our pivot this last week, as well as adding citations to a source we had used for information. We will also begin and finish work on our requirements document, in accordance with the guidelines specified on the class website.

8) Week 4: Eric Winkler: This Week:

This week we spent a chunk of time revisiting our problem statement and started a plan for the rest of the term. We set up a schedule for team meetings which will be weekly. We also spent a good chunk of time discussing basic requirements and touched on the design of our tool. We were mostly attempting to reassure ourselves that our project was feasible and made sense. The result of the discussion was a narrowing of our project's scope.

Next Week:

We're planning on having a completed(ish) version of our requirements done by the end of next week. Additionally we may spend a bit more time familiarizing ourselves with visual studio code its extension system.

9) Week 5: Cramer Smith: What We Did This Week

This week we worked on our problem statement final version. We only needed to add citation so it wasn't that bad. We also made some progress on our first HelloWorld Extension and making it look at the current files for the user. We also started our requirement document.

Our Plan for Next Week

Next we plan to do more of our requirements document and make a lot more progress on our simple prototype.

10) Week 5: Sam Lichlyter: This week we primarily worked on revising our Problem Statement as well as the rough draft of our Requirements Document. We came up with a tentative schedule for what we wanted to accomplish and when. Because we weren't given much direction in class as far as when certain documents are due, we had to more or less make it up ourselves. The schedule we came up with I think will allow us to move forward at a steady pace accomplishing the tasks and getting a working product by the end of Winter Term. This will give us plenty of time in the Spring to test our product with other people.

Next week we plan on getting some functionality built into our extension for VSCode.

11) Week 5: Zach Schneider: What We Did This Week

This week the team completed the final version of our problem stated. We added a citations, subject headings, and revised a few lines concerning how we will test our software tool. We also began work on our requirements document. A rough draft has been submitted as of Friday, with the majority of the required sections within the document completed. We still need to nail down some specifics regarding the actual functions within our tool, as well as the operations the user will perform within our tool.

We met with Vee this week and simply checked in our progress with him. There was nothing specific we needed to do coming away from that meeting, other than making general progress on our documents.

Plans For Next Week

We will continue to iterate through versions of our requirements document, completing and signing the final version by Friday 11/4. We will meet as a team to discuss the details still needed, and we will await and address and corrections as noted by the professors for this class.

12) Week 5: Eric Winkler: This week the majority of our time was spent writing the requirements document. Part of that process involved getting a very, very rough schedule for our project. We had to mostly make guesses as to how long each component would take but we left some flexibility room at the end. We also made some finishing touches to our problem statement.

The plan next week is to begin preparing for our next meeting with out client. He'd like to see something on the screen (even hello world as a VSC extension). We're planning on getting our windows set up.

13) Week 6: Cramer Smith: What We Did This Week

This week we worked on the Requirements document. While doing this we were forced to make a lot of decisions about the project that we have been needing to make. We were faced with a lot of requirements that we had to define that were previously undefined. I hope that we haven't set ourselves up for failure.

What We Plan To Do Next Week

With the next week we hope to get a small prototype done for our client, to prove that we have made progress in the development side. We also want to get some paper prototyping done so that we can get a better idea of what we want the

application to look like. We also need to do the tech document. I think that is what it is called.

14) Week 6: Sam Lichlyter: This week we finalized our requirements document. We met up a few times to finish the sections we hadn't finished last week and to revise the ones we had. Cramer and I worked on getting our hello world example to work with VSCode as well as resolve some issues Zach and Eric were having with GitHub and Windows path names being too long. We also met with our TA, Vee, to go over our progress. He also explained what a tech review was to us.

Next week we have another meeting with Chris. We plan on having a few paper prototypes (or derivatives thereof) to show him as well as our hello world extension.

15) Week 6: Zach Schneider: What We Did This Week

Our team continued to work on finishing our requirements document this week. We met twice to complete the remaining sections we had not finished by last Friday, then went through and revised ambiguous language and poor grammar. Sam and Cramer also continued to fiddle with our "hello world" extension in visual studio code. This test extension will likely transition into our proof of concept code that Chris asked for by mid November. Finally, we met with Ve to check our requirements progress with him and get some information about the tech review document that we will write next week.

Plans For Next Week

According to the CS461 schedule, we will begin writing our tech review document next week. This appears to be both an individual and team document. As we clarify what technology we plan to use, we will also begin to seriously work on our proof on concept visualization extension. We hope to have this p-o-c complete in two weeks.

16) Week 6: Eric Winkler: This week we worked towards finishing our requirements document. We also spent more time discussing design options and what facets of this project were feasible with the limited time we have to implement before spring testing. We also discussed and began work on a prototype for our client that will need to be complete next week.

Next week we will be working on the above mentioned prototype and sitting down with our client to discuss the project's progress.

17) Week 7: Cramer Smith: What We Did

This week we focused on getting out a very simple first prototype for our client. We made a extension that displays all the files in the users work station. We also worked on splitting up the project into 4 main parts that each of us can 'specialize' in. We plan on working on these different parts and each of us splitting them up even more while writing the tech document.

The four parts are:

Graphic User Interphase

Files

Parser

Integrated Development Environment

We are still trying to figure out who is going to be a in charge of which part except Sam is probably going to use the parser.

What We Are Going To Do

The next week we are going to get our Tech document assignment done, do some research into other tools that might possibly be better for our project.

18) Week 7: Sam Lichlyter: This week I helped to create our "Hello World" extension in Visual Studio Code that would parse the directory in which the user was currently working and create an HTML document that listed all the files in their working directory. Our team also figured out how to take that HTML document and display it inline in the editor. We started working on it using JavaScript but we ultimately decided to go with TypeScript, which is Microsofts version of JavaScript,

because almost all the examples of extensions used it. In the next few days we plan on finishing up our Technology Review. We have mostly been working in Visual Studio Code and JavaScript, but I think we should explore other options including other text editors such as Atom or Brackets. They are both similarly open sourced text editors.

19) Week 7: Zach Schneider: What We Did This Week

Each team member worked towards producing an initial prototype extension for VS Code to display all subfolders and files in a project directory. This ”Hello World” example was shown to our client during a meeting with him Thursday. Chris seemed satisfied with our progress so far, but also gave a few suggestions as to what direction each team member should take going forward. My primary responsibility for now will be to research and start developing a method to serialize and store parsed project file data within some kind of database. This will make it so our project parser does not have to re-parse the entire code project every time a file is changed. This research goes hand in hand with the technology review document we started this week. Each member has divided up various areas of interest/technologies to research and report on for use in our project. We expect to finish this document over the weekend.

Plans For Next Week

Aside from finishing the tech review document, we will begin to take a hard look at the overall design of our extension. We will continue to meet and prototype as a team to get a better idea of how we want to implement various features, and will compile these decisions in our design document that is due finals week.

20) Week 7: Eric Winkler: This week we spent a lot of time discussing design and prototyping.

We started to really nail down the design of our ’file map’. Ideally, it will look like a big web of files that we can zoom into but should the zooming prove too difficult, we also have an alternate design set up. We’re a little worried the file map might be overwhelming in terms of information being displayed to the user but otherwise we’re happy with the direction we’re planning to take.

Additionally, we spent a good chunk of time prototyping a visual studio code in preparation for a meeting we had on Thursday. We built an application that basically parsed for all the files in the current directory and returned it as html within a separate column in vs code. This got us over A LOT of technical humps we we’re worried with.

Our meeting with Chris went well. He gave us a pretty elegant architecture to set up our extension with which naturally divided into four big tasks we could each take responsibility for.

gg

21) Week 8: Cramer Smith: What We Did This Week

We spent a lot of time working on the tech document. I specifically worked on the sections:

IDE: looked at the Brackets, Atom, and VSCode

Language: looked at TypeScript and Javascript

Event Handling: looked at htmlpreview, event emitters, and the built in events.

I learned a lot about Visual Studio Code which was really nice I feel like I don’t know a lot about VSCode.

What we plan on doing next week

We plan to start our design doc ASAP, and work a little more on our prototypes.

22) Week 8: Sam Lichlyter: This week we finished up our Tech Review document and figured out a lot of the technologies we will be using throughout our project. The main piece I will be in charge of for our product is the file parser. I learned through our tech review that using regular expressions, which was my original plan, probably won’t be enough to parse through real world HTML code. For this reason I have decided we will need to write a Perl script and use some of the already built HTML parser modules to find the tags that we want to check for. One of the Perl parsers will also allow us to check against the W3C

standards so we can flag the users code for where they essentially ”broke the rules.” This module will also allow us to turn it off and check against how most browsers actually render the code since how the browser renders the markup and how the W3C specifies it should be rendered are sometimes different.

23) Week 8: Zach Schneider: What We Did This Week

This week the team spent most of our time researching technologies and writing the tech review document. Due to some questions that were brought up from some of our research, we request a one day extension for the document, as we wanted to discuss them before submitting our final conclusions. My role in the document was related to data serialization, as parsed data from our clients' project would need to be stored temporarily in some form. We decided that form would be JSON, and we would use the JSON.stringify function to serialize our parser data. Once the document was complete, we spent the rest of our time laying out the basic framework for our design document.

Plans for Next Week

Next week, we plan to begin writing our design document or our progress report. Due to the short week, we're not sure what work will be completed before the break.

24) Week 8: Eric Winkler: This Week we spent time discussing what our shared data structure will look like moving forward. This data structure will primarily be used by Sam and myself; Sam gathering the data and me using it.

Additionally we spent some time finishing up our tech review. We discussed the options each of use researched and made a few decisions about what technologies we'll be using. As far as I'm aware, we are still sticking with VS code and Typescript as our primary technologies, and using a JSON file for data storage.

Next week we should be doing a bit more prototyping and working on our presentation and poster board.

25) Week 9: Cramer Smith: What I did This Week

Not a whole lot. I did look around the documentation for VSCode and started to think about the design document. That is truly what spurred the curiosity to look further into the specific documentation of VSCode. I am worried we won't know enough to write a fully fledged design document. I hope to look into this more during the break and will try to do some programming this week.

Plan for Next Week

We plan to start and finish the design document, and work a lot on our actua project. I feel as though working on the design document should fuel the work on the actual program.

26) Week 9: Sam Lichlyter: This week being a long holiday weekend I didn't work on a whole lot beforehand. I do plan on looking over everything we need for the parser and how to build one. Also at a more basic level I plan on looking over some Perl tutorials to get a better handle on the language.

Next week we plan on working on our Design Document and finishing it up as well as another meeting with our client.

27) Week 9: Zach Schneider: What We Did This Week

This week the team discussed our schedule for completing the design document, the progress report, and what work we want to accomplish on our extension for the rest of the term. I also researched options for the visual portion of our progress report–I opted for OBS and PowerPoint as I am most familiar with those tools. Aside from this, nothing else was done due to the short week.

Plans For Next Week

Next week we will begin writing our design document. We will continue work on the prototype extension with the goal of having another feature done before our meeting with our client on Friday.

28) Week 9: Eric Winkler: This week we haven't yet committed much time to the project. The plan is to begin work on a first version of the file map within the next couple of days. The goals of this first version will be to read in test data from a build of our current shared data structure and render them as objects with Visual Studio code. We are planning on starting off with vis.js and moving through our other options until we find something that will work within the VS code environment.

29) Week 10: Cramer Smith: What we did this week

This week we made a design document. This made us look more into the details of our project and 'how' it was going to work. We also met with our client and showed him our most recent prototype. I feel like he was not happy.

What we plan todo next week

nothing have a nice break during finals, then during break we will work on getting the prototype working way better.

30) Week 10: Sam Lichlyter: This week we finished up our Design Document. We also had a meeting with our client in which we talked about the progress we made since our last meeting as well as our plan for what we want to get done over the Christmas break. I also messed with our parser a bit make it better at finding what we wanted.

31) Week 10: Zach Schneider: What We Did This Week

The entirety of this week was spent completing the design document. We all met earlier in the week to define what each team member would write about. We each tackled the general scope of the components we discussed in our Tech Review last week. I covered the data handling portion, that is, the data structure, serialization of the data, and storage of the data into a JSON file. These components were introduced and discussed in light of their respective design viewpoints, according to the IEEE 1016-2009 document. Additionally, we met with our client Chris to discuss the progress we've made thus far, as well as our plan over winter break.

Plans For Next Week

During finals week, we plan to write our progress reports for the term and create a verbal presentation of those reports. We are confident that the report will be thoroughly completed by the due date on Wednesday. Post finals week, we will continue to work on our VS Code extension over the break, and plan to have a rough prototype with parsing and UI completed by the beginning of Winter term.

32) Week 10: Eric Winkler: This week we primarily worked on building out our design document. This was a lot hard than the previous documents as we had a bit of trouble trying to interpreter the IEEE standard. It was fairly vague in it's explanations. We later found out during our TA meeting that that was by design as a highly specific document might place constraints on a projects design. We also had a brief meeting with our client to discuss plans and deadlines for over break.

Next week we will be primarily working on our final report for the term and making plans for the next few weeks.

33) Week 1: Cramer Smith: What We Did

This week we met with our Client and showed him our first prototype. It was not to the point that we wanted but he seemed very happy with the results and gave us some good feedback and advice for the future.

We did a lot of work toward getting our extension into an actual working product. Previously we had three different parts and now they are finally coming together. But the bring these parts together was more work than we initially thought it would be. There was a lot of debugging and a lot of frustration, but we are quickly getting much closer to our end goal.

What We Plan to Do

This next week we are going to crank it down a notch. We don't have a deadline from our client until the end of february and we should be able to work more slow and steady rather than this weeks full pace non stop. But we will keep working on it.

B. Winter Term

1) Week 1: Sam Lichlyter: Winter Break

Over winter break we tried to get a lot of the development done while we had some extra free time. The UI logic that Eric was working on is pretty close to being finished, I worked a bit on the parsers over break but didn't make a ton of headway until this week actually.

This Week

Like I said, this week I made significant progress on the parsers. I worked mainly on getting data from each of the files we were parsing and Zach and Cramer worked on taking the data I had found and serializing it into a JSON data structure that our UI could read. Before we had pretty separate pieces all doing different things, but this week we put them all together so they could all communicate which tied up our project pretty nicely.

We also met with our client this week to catch up on our progress over the break and decided where we would go from here.

Moving Forward

Looking to the rest of the term we still have a few feature we need to implement like getting the links between our nodes to show up, I know we're actually really close on this. Our client gave us some pretty good advice about hooking up our parser logic to the UI and how our users should interact with our tool. We also talked about timelines and what milestones we should aim for within the next couple of months.

2) Week 1: Zach Schneider: What We Did Over Break and Week 1:

As of this week, our team is in a pretty good spot concerning work we've done so far and goals we're working on currently and in the future. Sam, Eric and I are all from the same town so we were able to meet up briefly at the beginning of break to determine what we wanted to accomplish. We essentially decided that we wanted each separate piece of our project to basically be in a working prototype state by the end of Week 1 of Winter Term. Each member of our team had previously undertaken a general concern/category within the project. We informed our other team member, Cramer, of our plans following the meeting. We then each made separate progress over the rest of the break. The UI, the data layer/transfer, and the file parser reached a working state during this time period.

The team had scheduled a meeting with Chris Scaffidi (our client) for the end of Week 1, and our general hope was that we could have some of our project pieces complete and put together by that time. Unfortunately, the ”assembly” process was much more time consuming than we initially anticipated, even after working a couple dozen hours between us during Week 1. Despite this, Chris was fairly happy with our progress so far and believed we were on track both time-wise and feature-wise. His goal for us was to have a prototype that we would be able to use by the end of February, and a prototype that other CS friends could use by the end of March. We believe these timelines align with Kevin's ”alpha” and ”beta” dates set on the course website.

Plans For This Week:

This week, we continue to make progress on each respective feature or function that seems the most relatively crucial to the project. I'm continuing to flesh out the parser-¿UI data layer so that more parser data can be transferred. We're in a spot where the whole team can work together towards commons goals, rather than individual ones as before break.

3) Week 1: Eric Winkler: Over break, the Postal team began implementation. We had three primary modules that were worked on be different people. Sam worked primarily on the parser, Zach on the JSON implmentation and access methods and myself on the UI.

Over break I created the UI within an electron application. The UI is currently capable of reading in our standardized JSON and generating an interactive file map. The file map nodes are color coded by file type, have a size determined by the number of links to the node, can be dragged around and feature a mostly complete implementation of the ”dig down” functionality. The file map can also be zoomed and panned.

The next step is to update the file map to be generated on a command from VSCode. We’re currently having a little trouble with this as it appears that electron needs to be launched in a certain way from its own process.

WE also had a meeting with our client this first week. We brought him up to speed with where the project is and what was giving the team difficulty. He offered some very good advice about how to further develop the tool. He specifically said to focus on the one feature that makes our tool most useful and to think about what situation it might be most beneficial for testing purposes.

4) Week 2: Cramer Smith: What we did this week

This week we met and talked out some of the progress we are having. We also assigned some more work to me.

What we are planning to do

In the next week I hope to work on the Error handling and the UI that the user will see.

5) Week 2: Sam Lichlyter: This week

This week we met up to go over the progress we’d made since last week. Eric made some progress on the UI, Cramer made some progress on getting the editor to show the errors we find, Zach and I worked on the parser a bit to get our links working that didn’t work last week. Zach and I are pretty close are getting the links to work as well as passing more data to the UI.

Next Week

Next week I would like to get the UI to display nodes for directories we find in the main directory we’re parsing.

6) Week 2: Zach Schneider: What We Did This Week:

This week the team met to see where our project stood, what functioned and what needed to function next. Eric continued to make progress on the Electron UI which displays each file in the project, while I continue to work on created the links between those files. By the end of the week, links between files were displayed visually in the electron UI. However, there most likely will need to be a rework of this functionality: the links are generated independently of the project files, which causes major id number incongruities for larger web project when parsed.

Plans For Next Week:

Going into the next week, I will work on reworking the link generation system in the parser. This may or may not be a large effort, but it may go more smoothly if I work with Sam to get the file data I need from the parser. Meanwhile, Cramer and Eric are starting to working on triggering the parser by a save operation within VS Code. Eric will have the UI update accordingly at the same time as the save action.

7) Week 2: Eric Winkler: This week we met to discuss some bugs with the extension when it parsed larger projects. The leading theory was that the parser would create a JSON that had links to node ids that didn’t have their own structure in the JSON. This caused the GUI to try to link nodes that might not have existed. Otherwise, the extensions performance seems to work well on larger (+100 nodes) projects so far.

The plan for the weekend is to get the ”dig down” functionality refactored. I’ll probably be adding a couple buttons to help control that.

8) Week 3: Cramer Smith: what I did this week

This week I didn’t really do too much. I started working on my error handling. It doesnt quite work but it does highlight a line all red.

what I plan to do next week

This next week I hope to get it fully working and start work on my error UI. And figure out how to trigger the extension on save.

9) Week 3: Sam Lichlyter: This week

This week I didn't get as much done as I was planning. I made some progress on the parser but didn't get to working on the links or getting the directories to show up as nodes. Our team also met up and decided we should move our creation of the data structure to the parser instead of parsing then creating the data structure. This would save some overhead of passing data around and would prevent us from touching the data twice.

This will be my task for next week. We also met with Vee for the first time this term to catch up and go over what we would be doing this term for the class, apart from working on our project.

10) Week 3: Zach Schneider: What We Did This Week

This week was a relatively slow week for most of the team. We met with Vee on Monday to discuss where we were on our progress and then met as a team afterward to see where each member was on their individual tasks. We are all content with the progress we've made so far, but all agree that February will require a bit more work time put in. We still aim to have a fully functional prototype by the end of February.

Plans For Next Week

Next week should see significantly more work done, as we're all pretty busy with other concerns this week and on the weekend. I plan to rework the creation of file nodes so that links will be correctly supported for Eric's UI, and I will work with Sam and the file parser to ensure this is done sustainably.

11) Week 3: Eric Winkler: This week I mostly focused on getting our electron/node GUI to launch when it is called from the vs code extension. It's not quite working yet, but I believe I'm getting close. I've been looking into a node module called process-bridge which should help out with this problem.

I hope to be done with this next week and also start on the dig down refactor and styling the window.

12) Week 4: Cramer Smith: I worked on the error notification system

I plan on working on the on save event, and continued development on the error UI.

13) Week 4: Sam Lichlyter: This week

This week I got a little more time to work on the parser than last week, but I still didn't get the directories to show up. I think I will hand this off to Zach because he seems to be working more with this data than I am at the moment. I'm going to move on for now and switch our program to a more defined Model View Controller design pattern. Eric has started creating a controller, I just need to refactor the code we wrote for the VSCode extension itself.

Next Week

Moving our extension to MVC might take a little longer, so I will also work on this next week.

14) Week 4: Zach Schneider: What We Did This Week

This week the team met up and made some progress on each of the pieces we've tasked each other with going forward. Eric successfully launched our Electron UI window from the extension itself instead of the command line, serving as a major step forward for the UI. I continue to rework the Data Structure file node generation, which is conceptually proving more difficult than expected. I will work with Sam going forward to make sure the parser gives me the data I need. Cramer successfully got code highlighting working, a step in the right direction for our error management. We also had class this week which informed us of our upcoming progress report and OneNote tasks.

Plans For Next Week

I will set up OneNote for our ”Engineering Notebook” and begin to compile the information we need to make our midterm progress report. I will also continue working on fixing Data Structure file node generation.

15) Week 4: Eric Winkler: This week I finally got the GUI to launch from the extension pragmatically! this took quite a bit longer than I thought it would (it shold not have been this hard...) I didn't actually need to use the process bridge to get it to launch. Instead I found the electron has an executable buried way, way down in its npm package directory. I can launch that from a new node thread and everything seems to work. It also works on the three OSs we're concerned about.

The process bridge wasn't a loss as we'll probably need it for communications between the two node processes.

16) Week 5: Cramer Smith: What I did this Week.

I made a function that does a regex function over a file line by line

What I plan to do next week

The next week I plan to work on the details of getting our extension on the 'store' and getting it out to the people.

17) Week 5: Sam Lichlyter: This Week

I started to move our extension to an MVC framework. To do this I basically took a lot of the functionality we had and ”functionalized” our code into a controller class. This will also help Zach out with his asynchronous issues he's having so he can just call one of these functions when he needs to instead of having to other workarounds.

Next week

Next week I will continue moving things over to MVC.

18) Week 5: Zach Schneider: What We Did This Week

This week I began to wrap up node creation in the data structure portion of our project. The display of href links between nodes has continued to be tricky, and isn't fully featured or functional yet, but I believe this can be added to over time. The rest of the team worked on their respective portions as well.

Plans for Next Week

Once we are in a good place concerning the actual project, the team will convene to work on revising the tech review and design document. Tech review should be short and sweet, while the design doc may take some serious reevaluation. Kevin allowed us an extension for our progress report in light of our attempt at a beta release by Week 7 instead of 11. The progress report will be completed by the end of Week 7 instead.

19) Week 5: Eric Winkler: We spent this last week developing and working towards a plan to provide a public release of Postal on the extension marketplace. This release will not be entirely feature complete but we'd like to have feedback on what we currently have working by the end of the month. Our plan is to finish all functionality short of errors, release what we have on Monday (2/20) and during that week finish errors and solicit feed back from both the market place and personal connections.

I will be focusing on cleaning up the UI and getting the nodes in a hierarchy arrangement. We came to the conclusion that displacing both links in terms of includes and hyperlinks as well as displaying links that indicate a node's position in the project directory was overwhelming. This is especially true in a web like configuration of nodes. Instead, we will display the nodes in a directory based hierarchy where the links displayed by default will be indicative of the node's position in the project directory. When a node is clicked on, the links that indicate a connect between files (includes/hyperlinks) will then be rendered.

20) Week 6: Cramer Smith: What we did this week

This week we went over our old documentation and I made a line by line parser that we now don't need. That's ok.

What we plan to do next week

In the next week we plan to get our 1.0.0 out, and then we will make video of our mid way point.

21) Week 6: Sam Lichlyter: This Week

This week we updated a lot of our documents as per requested by Kevin including the Tech Review and the Design Document. This was good because enough had changed in our design that our old design document was out of date. This weekend, we sat down and refactored our entire extension. It is now more reliable and actually has more features than it had before we started this weekend.

Next Week

Next week we will work on our Progress Report because it was extended a week by Kevin for us because we wanted to get our extension on the VSCode Extension Marketplace as soon as possible. We will also work on uploading our extension to the marketplace and finishing up the refactors by tomorrow evening.

22) Week 6: Zach Schneider: What We Did This Week

This week the team spent most of our time working on revising our various documents. Minor revisions were made to update our design document, while some serious portions of the tech review had to be updated. I was in charge of setting up the One Note pages and formatting them correctly with all the needed content for our midterm progress report. Lastly, we scheduled a team code review for this weekend so each member can understand what everyone else is working on. Hopefully this leads to cleaner and more functional code overall. We plan to work a lot this weekend in order to reach our Beta release by next week so we can begin user testing.

Plans For Next Week

Next week we will finish our progress report presentation, as we received an extension on it. The extension is due to our accelerated work timeline for our actual project. If all major features are completed by the end of the week, we will begin working on implementing error identification and visualization for our user's projects.

23) Week 6: Eric Winkler: This week we began the process of of refactoring all of postal. It actually turned into a complete redesign and rebuild from scratch. I mostly headed the redesign while Sam became our Code-driver. As a result there are now clear Interfaces between our Parser, Controller and UI and all of our code is of a much higher quality. I will be doing a design document update to reflect these changes, they are a little too numerous to mention in a blog post. At a high level the extension now works like this:

The user luanches the Postal extension.

Our extension calls our controller which then:

Finds all files and directories recursively.

Calls the parser to parse each file. This returns an array of standardized Tokens.

The controller turns those tokens into our JSON File and calls the UI.

The UI reads the JSON and launches.

24) Week 7: Cramer Smith: what we did This week

This week we completely redid the extension. We made a lot of improvements to the prose and the th way that the graph is generated in the UI. I dont know how it works but it seems to work really well

We also put the extension up on the market place. This is great cause now we can distribute our extension much easier.

what we plan to do this week

I would like to get a some settings done for our extension. done w

25) Week 7: Sam Lichlyter: This Week

This week we continued to work on our extension. The biggest thing was we uploaded it to the Microsoft Visual Studio Code Marketplace so we could get users feedback on it. So far we haven't heard from any users on the store but we have given it to a few of our friends to test it out for us. We fixed some bugs related to the store and some general ones we didn't realize were there until we gave it to people.

We also put together our progress report this week.

Next Week

Next week we have a meeting with our client and we also plan on pushing some more bug fixes to the store.

26) Week 7: Zach Schneider: What We Did This Week

This last week was extremely busy. The team worked all weekend, essentially re-writing each component of our project, piece by piece so that they would support the required design specifications. Previously we had had a mostly working prototype, but could not build on it any further. The parser saw the most work re done, followed by the UI then the data layer. Ultimately, we achieved the milestone we wanted and published our extension to the VS Code extension marketplace, where users can now download and use it. Additionally, we completed our progress report, which we had received an extension on last week.

Plans For Next Week

Next week we will meet with our client, Prof Scaffidi to show him our progress and find out about the research component of this project. We will continue to test, bug fix and add features to the extension, particularly around error detection in a user's code. We also have class next week to learn about how to present our project at Expo.

27) Week 7: Eric Winkler: This week we finished our refactor of Postal and implemented a few new features (our dig down is currently working better than before). We also published it on the store on Monday and have been bug fixing since. We still have to implement the one click links appearing and error parsing.

A challenge we have is that our npm modules are too big to include packaged in our extension, so we have to currently have the user run 'npm install' in their extension directory before it will work. This is not ideal and we're currently looking for ways around this.

28) Week 8: Cramer Smith: What I Did This Week

Well we met with chris. Was that this week? Yeah it was, and he wasn't as happy as we had hoped. I did get a simple start on setting though and I am happy about that, I don't know if we will use them.

What I plan On Doing This Week

This week we plan on making the extension more useful. We have a huge list of features that we plan to add this weekend.

29) Week 8: Sam Lichlyter: This Week

This week we met with our client to show him our progress on our extension. He gave us some valuable feedback on the UI of our extension. We went on to try and implement these as well as fix some of the bugs we found last week.

Next Week

Next week we plan on getting c-like functions going.

30) Week 8: Zach Schneider: What We Did This Week

We started the week of by meeting with or client, Chris Scaffidi. While he was happy about our progress, he still felt our project/UI wasn't to the stage of usefulness yet where someone (or us) would want to use it every day. So, our goal going forward is to move from the core visualization functionality, to making that visualization useful. This will include letting users

jump to the point in code that each node represent and better visibility with node names and relationship. We will convene again this weekend to hash out much of the work that needs to be done.

Plans for Next Week

We will evaluate where the project is after our work/code review this weekend. We hope to solidify the rest of the core functionality of the parser and begin adding ”gold plating” features (though at this point those features are more or less necessary). Our new major deadline is the end of the term when we hope to have our project in a ”useful” state.

31) Week 8: Eric Winkler: This week we met with our client and decided on a list of features to complete in the next couple weeks. this includes refactoring our UI with a new vis method called clustering. Additionally we will be implementing C-like function parsing and other UI features like being able to switch between web view and a hierarchy view.

32) Week 9: Cramer Smith: What we did this week.

Almost nothing. We worked some this weekend we now have C functions working. That makes our parser way more useful to me personally.

What we plan to do next week

Next week there are a number of things that we need to do. We have to do our progress report again and make a video thing for that. We also hope to have our project entirely done by the end of this quarter so we will probably work on the final touches of our extension and get it up on the marketplace and see if we can get some actual users.

33) Week 9: Sam Lichlyter: This Week

This week we implemented c-like functions. This allows our tool to visualize c-like functions similar to the way we visualize divs and links in our web projects. We also tried to implement a character-by-character parser, but we decided we could rework our current parser to parse the whole file at once and still get the same features we wanted out of a character-by-character parser. We also started on clustering our nodes.

Next Week

Next week we plan on getting our clustering to completely work as well as some other basic features like navigating to the users code when they click on links, not parsing the user’s comments, and getting a .postalignore working so they can specify which files they don’t want our extension to parse.

34) Week 9: Zach Schneider: What We Did This Week

This week each team member spent most of their time on individual work on adding features to the project. I worked on adding configuration buttons to the UI that allow the user to dynamically change the shape and physics of our information network. Later, I began working on the UI for the list of errors that our parsers will detect in each file. Lastly, the team began making preparations for our progress report video and preliminary poster due next week.

Plans For Next Week

Next week we will meet to put together the basic version of our Expo poster, deciding what points we want to hit and how we want to display our UI. Additionally, we will continue adding minor features to our project, such as the error list.

35) Week 9: Eric Winkler: This week we planned on implementing a character by character parser but instead opted for a sort of priority base parser that reads in the entire file. This allow us to write grammars for things that may appear on multiple lines like for c-function definitions. Additionally we began the process of switching over to a clustering method for our UI nodes and Zach worked on UI controls.

36) Week 10: Cramer Smith: What We Did

We continued work on the large todo list that we feel we need to have finished by the time of actual release, and we are making progress. I specifically working on the process bridge that will connect the electron terminal to the Vscode IDE.

What We Plan To Do

With Spring break finally here I think some of us are going to take a small break. I myself think I will try to get the process bridge working and have the users be able to navigate to the code from the electron window, but we will see if I will be able to accomplish that.

37) Week 10: Sam Lichlyter: This Week

This week I fixed our postalignore to allow a set of files or directories to ignore by the parser. I also moved it the settings that the user can edit within VSCode. This is instead of a separate file they would have to include. I also built a new grammar and parser behavior to allow comments to be ignored. This fixed some issues where ‘if‘ and ‘while‘ blocks were being considered functions by our default grammars.

38) Week 10: Zach Schneider: What We Did This Week

We had two major goals we worked on this week: completing our preliminary Expo poster and finishing a number of UI features for our project. The poster had a bit of a rough start; we all did a part of the poster separately, then later realized that the whole thing was too wordy and not cohesive. We revamped the poster as a team, focusing more on why we did what we did rather than including so many technical details on how we did it. For the project itself, I worked on adding a sidebar window that will display an notifications or errors the parser finds for each node in the main UI. The visual elements are there, it’s just down to getting the parser to generate the right data to populate the window.

Plans For Next Week

Next week we will each work on our individual progress reports as well as the team video report. It should be mostly the same as the one we did a few weeks ago for the midterm. We may make minor updates to the codebase as well.

39) Week 10: Eric Winkler: This last week we began to implement the last few features for a close to finished version of postal. I worked a bit on fixings some bugs with the new node clustering system. Additionally I added the ability to see link information (line numbers) when links are selected. This was harder to do than I originally anticipated. The clustering system actually creates all new edges and nodes when things become clustered, so finding the correct objects to manipulate was a bit of a pain. Additionally I reintroduced file links as red lines but they will be changed again here relatively quickly.

Next I will be working on adding options for displaying file links, making sure our process bridge is working correctly and adding a couple of UI controls.

C. Spring Term

1) Week 1: Cramer Smith: What We Did Last Week

We had spring break and we all worked somewhat sporadically over the break, not that much though. Then we got back to school and we met on Monday and worked on some bugs and features that needed to be added. We got those to a working stage and then we met with our client on Tuesday. This meeting went poorly. He really wanted progress on the UI and we haven’t been focusing on the UI as much as the functionality. That was disappointing, but with what we have set up already we feel we can make the changes he wants to see by our next meeting. So those changes are what we have been working on.

What We Plan To Do

This next week we meet with our client and hopefully show him a better UI. If that meeting goes well he will start gathering funds for testing and get us started on making good experiments for users. We also will continue to squash the numerous bugs

that we are running into with the extension.

2) Week 1: Sam Lichlyter: This Week

This week we met with our client who was slightly disappointed in the progress we had made in the last few weeks but offered valuable feedback and a chance for us to try again next week. So this week we have focused primarily on the feedback given to us by our client so we can meet his expectations by our next meeting. I have mostly been in charge of changing some of the subtler UI elements to make the tool more user friends and a bit easier to find/navigate information.

3) Week 1: Zach Schneider: What We Did This Week

This week we met with out client to check in with our progress. We had a bit of a misunderstanding of work focus: we had been polishing off features whereas he wanted us to focus on polishing the UI. As such, we will meet again next week to check our progress again. Various features were worked on this week, mostly related to the UI. We created a number of event listeners that allow users to click on our node network and be directed to the corresponding line of code in their project. Additionally, we created a TypeScript grammar that will allow us to visualize our own project, as it is the main coding project we actually all work on day to day.

Plans for Next Week

We will continue polishing the UI to get it up to our client's expectations. We have a meeting with him on Tuesday to see if the work is to his liking. After that, we will see if there is time/resources available to conduct user testing.

4) Week 1: Eric Winkler: The week We implemented a few new features and had a meeting with our Client.

I spent a bit of time working on fixing some performance problems with vis.js. When running in a hierarchical mode, vis runs several optimization algorithms in an attempt to limit white space between nodes. However, our trees aren't perfect and can nodes can have links over several levels of the tree and this seems to cause vis to hang when it tries to optimize. I'm currently midway through trying to massage the input such that vis will begin to behave in a more useful way.

The result of our client meeting was that we would need to work on the UI a bitfocus on using the tool daily. We're planning on meeting with him again next week to show our progress.

5) Week 2: Cramer Smith: What We Did This Week

This week we worked on improving the UI for our client. We made it more responsive and improved the quality of information that is produced and shown on the UI. But with all this improvement we also introduced a number of bugs that will need to be addressed in the future. But with the improvements we showed our client we were able to get him satisfied with what we had and he said that we should start organizing a testing group that will try the extension and that we can get results from the tests and maybe publish the work. This meant we had to get certified by the IRB, so we worked on getting certified and working on creating a short questionnaire that we will use to quantify the results of the tests.

What we plan to do this week

This week we hope to address some of the more pressing bugs and create two use cases that the people we will be experimenting on will be using during the test.

6) Week 2: Sam Lichlyter: This Week

This week we worked on fixing minor bugs within our tool. For example we had some UI that was a little wonky that we fixed up, we also still have a bug that catches if, else, while, etc. blocks as functions when defined in TypeScript. But mostly minor bugs like this were taken care of this week. We also met with our client again to to talk about IRB testing. This went well and we have a plan to move forward in the coming weeks to get testers.

Next Week

Next week we plan on fixing smaller bugs for our tool, but I think mostly our focus will be on the poster board that's coming due soon.

7) Week 2: Zach Schneider: What We Did This Week

This week we finished up most of the UI changes we felt Chris cared about most (font, node hierarchy, readable notifications) and demoed it to him on Tuesday. He was satisfied with our progress and began the discussions about user testing. That will be our main focus outside of Expo going forward. We are writing questions to ask the users once they have tested our project that we will submit to the OSU IRB Monday. Additionally, we worked on our Capstone poster second draft. We are planning on capitalizing on the extra credit opportunity concerning the poster board by working with another team.

Plans for Next Week

Next week we will begin creating the tasks the users will perform in our study. We will also continue polishing our project to make sure it's totally ready. We will refer to Chris to make sure our study is sound.

8) Week 2: Eric Winkler: This week we continued to adjust postal to better fit our client's needs in anticipation of a meeting we had on Tuesday. He was happy with the improvements we had made and was also pleased that we were using the tool on our own time. We began to make plans for testing and created a new timeline for the group. In the next few days the team will get IRB certified and create a list of questions that will eventually become our end of test questionnaire. I personally will be working on a couple of bugs I noticed as well.

9) Week 3: Cramer Smith: What We Did This Week

This week we worked on some minor bugs, we met with Kirsten about our poster, and we started working on getting demo projects for our user testing. This was difficult because we had to make two projects that were similar but not so similar that they had problems that were in the same or recognizable spots. We need the first test to not teach the user information of the second test. The Poster feedback we got was that we need less (more concise) text, and the pictures need to be a bit bigger. There were also some nitpicky changes that we hope to change.

What We Plan To Do Next Week

This week we hope to make all the improvements on the poster and the get some bugs fixes for the extension and finish the demo projects and maybe even get some our friends to unofficial run through our tests and questionnaire to get an idea of how the tests run and how they feel to user and for us as testers, but we will see if we get that far.

10) Week 3: Sam Lichlyter: This Week

This week we met with Kirsten to review our poster and another group's poster. We also started working on our user tests for our IRB testing. We also finished all of our IRB certification this week. We didn't really run into any problems other than things we needed to fix on our poster which we just got that feedback today.

Next Week

Next week we plan on continuing working on our user test.

11) Week 3: Zach Schneider: What We Did This Week

This week we submitted our user testing proposal off for IRB approval via Chris. This proposal included the questions we planned to ask our users and our own Human Subject certifications. Additionally, we further tweaked our project to make it completely testing ready. The team discussed the nature of the tasks we will ask users to carry out in order to test our project. These tasks should be finalized by next week. Lastly, we met as a team with Kirsten to receive feedback on our Expo poster and how we can explain our project more clearly.

Plans For Next Week

Next week we will make modifications to our poster according to the feedback we received, then send it off to Chris for final approval before Expo submission. Secondly, we plan to finalize our user testing plans as we wait for IRB approval to come back to us. Lastly, we'll check if there's any word on whether we have to do a midterm progress report again this term. If so, we will start on that.

12) Week 3: Eric Winkler: This week the team continued forward with our plans to prepare for testing. Our client has a hold of our certificates for IRB approval and our list of questions. He suggested that we begin prototyping the trials for the test and have a rough draft of the experiment by next Monday. We plan on completing this this weekend.

I also had a quick meeting with Eugene Zhang looking for feedback on our UI. He has some pretty good ideas as to how to make the UI a bit more intuitive and also put me in contact with Yue Zhang. I will be meeting with her next week to get some feedback on the project as well.

This weekend I also plan on doing a redesign of our poster as a result of a feedback meeting we had today.

13) Week 4: Cramer Smith: What We Did This Week.

We had to finish up the bug-crunching and adding the last small features to our extension in expectation of the code freeze. That approaching this coming up on Monday. The major requirements that we had to complete were:

1. be able to parse a large project quickly. We haven't tested that.
2. Be able to hover over notification list and highlight node where it is located.
3. Be able to click on the list and navigate to the code area.

We were able to test the large project, but the tests need to be more extensive at this moment.

What We Plan To Do Next week

This weekend we plan on implementing the other two features and they should be very easy to implement given that all the ground work is in place. We are not worried about the code freeze except for proving that we can parse a large project really quickly.

14) Week 4: Sam Lichlyter: This Week

This week we worked on our poster board for expo mainly. We went through a few iterations working with Kirsten and our client. Eric was the main design guy and the rest of us worked on the content of the poster. We didn't run into many issues other than a few communication issues which were worked out.

Next Week

Next week I imagine will mainly be our WIRED writing assignment.

15) Week 4: Zach Schneider: What we did this week

This week the team finalized our expo poster and got client approval. It's been submitted to Kevin for a final check. The team also developed the tasks and questions we will use to have users test the effectiveness of our tool. We still await IRB approval.

Plans for next week

Next week will finalize any last changes before the code freeze. Additionally we will interview and carry out the the wired writing assignment for the class.

16) Week 4: Eric Winkler: This week I primarily worked on our poster board. Kirsten asked that I write up a list of design tips for other groups which she seemed to appreciate. Unfortunately, the design I worked on violated a couple a guidelines so I had to revert back to an older version and submit a modified version of that.

We also produced a basic list of questions that will eventually become our questionnaire for user testing. We sent the list to our client for suggestions.

Our plan is to make some last minute fixes this week and to also do a couple of test runs with people we know using the above mentioned questions we produced for our client.

17) Week 5: Cramer Smith: What We Did This Week

We finished all the features of the extension in anticipation of the code freeze on Monday. So we added 1) The highlighted nodes 2) the navigate to code on notification click. Besides that, we worked on a Wired Article for Senior Design. We wanted to get more work done on the test cases for our experiment but after the code freeze we lost some motivation and took a break.

What We Plan To Do this Week

We hope to finish up the use cases for our experiment than get some of our friends to try out the extension and run the user case study with them to test the test.

18) Week 5: Sam Lichlyter: This Week

This week we submitted our poster board for expo. We had to make a few last minute adjustments to clean it up a bit and make it easier to read. We also finished up everything for the code freeze. We implemented a few last minute features to make the tool a little bit easier as well as fixing some of the last remaining small bugs. We also interviewed for and wrote our WIRED article. I wrote mine for the STEM Academy Data Solutions team. I interviewed Shannon Ernst who seemed like the team lead. This week we also tried to find someone who would be willing to try out our tool so we could test our user tests, but we came up short.

Next Week

Next week we plan on looking for someone again to test out our user tests on. We also plan on starting our progress report which is due the following Monday.

19) Week 5: Zach Schneider: What We Did This Week

This week the team wrapped up all code commits by the Monday night code freeze. We confirmed that we met all requirements and had fixed most bugs. We pushed these changes to our VS Code extension on Microsoft's extension marketplace. We are happy with the state our project is in and look forward to getting user feedback. Additionally, I submitted the team's expo poster by the deadline.

Individually, I met up with my assigned WIRED article partner and interviewed him about his project. I reviewed his project in the style of a WIRED article and submitted it to Kirsten by Friday.

Plans for Next Week

Next week the team will get started on our midterm progress report. The only uncertainty left for us is when the IRB testing proposal will be approved and when we will conduct our user testing.

20) Week 5: Eric Winkler: This week we began finishing features in preparation for the code freeze. I focused on fixing a couple bugs and getting individual nodes to highlight when the user hovered over a notification in the notification list.

We also released a 1.0 on the VScode store which has a ton of improvements since our last push.

We also began to make some progress on user testing by finding pre-test subjects and creating a first draft of a test form.

We still have a lot of work to do creating the test environment. Our form calls for a couple projects for the users to run Postal on involving some pretty specific functionality. We're having a little trouble finding directories to use and may have to make some ourselves.

Next week we will continue developing our user test forms.

21) Week 6: Cramer Smith: What We Did This Week

This week we finished up our user test testing, and use cases. And we gave our extension to some people to try our testing. The Problem we had is that we should have probably had this done last week, we are behind schedule a bit. But it's weird cause we aren't worried about the class anymore. We know that we will be fine for Expo and all the write-ups that need to happen. But for our client, we are probably behind, but not enough to be worrisome.

What We Plan To Do Next Week

Getting the people to test our tests was really good we now know that we need to change and we will probably be adding the finishing touches to the tests and then ask our client for some feedback on the test and then we will hopefully survive expo. We will also be making our progress report.

22) Week 6: Sam Lichlyter: This Week

This week we sat down with a few people to go over our user tests before we went through the actual IRB approved tests. This way we could fix any mistakes we had in our tests. We also starting putting together our progress report and video. One thing we haven't done is nail down our expo pitch with needs to happen soon. Unfortunately we are pretty good at getting across what our extension does to other developers but not the average joe. Personally I'm also struggling on relating our extension to younger kids.

Next Week

Next week we are going to wrap up our progress report on Monday and prepare for expo the rest of the week. I have full confidence in our ability to be prepared for expo by Friday.

23) Week 6: Zach Schneider: What We Did This Week

This week the team started work on the progress report and our presentation video of the report. Because we have our previous reports to look back on, many of the problems and solutions we had in the past are easy to recall and report as resolved now. One difficulty we've having is narrowing down a unified pitch for Expo. Our project may make sense to other developers, but requires additional foreknowledge for the layman. We have to have that pitch nailed down this weekend. Additionally, we began testing our project on users (other computer science students we know) in preparation for our actual testing phase when we hear back from the IRB.

Plans Next Week

Next week we will wrap up our progress report and prepare to demo at Expo. I believe we will be fully ready by Friday to show off our years' work.

24) Week 6: Eric Winkler: This week we mostly focused on our user test development and our midterm review.

I sat down with a classmate on Friday and ran through both of our trail exams with him.

it was a very useful experience as I found out that we hadn't written grammars for one of the tests yet, Needed to double our number of questions and the tester had actually found a bug in our test as well. There were some other smaller issues as well, like one of the project directories not having any TODOs in them despite having questions about them.

For the next week we will continue tuning our tests, making the above changes. I will also be making a few fixes in our code as well, I may add a reset button at the suggestion of our tester.

25) Week 7: Cramer Smith: What I Would Do Different

We did a project! And now it is done. Looking back it was hard. If I were to change anything I would have told us how to design the thing prior to the attempted implementation. We had to redo the extension 3 times in a row to finally get it to the state that it is now. That was a giant headache that should have been avoided but sadly was not. It is hard to say that if we had had more time to design in the very beginning that we would have had more success in that specific implementation detail of making the parser, but I think it could have helped us figure out problems that we would have run into before hand.

What I learned

I think that the biggest thing that I learned about myself during this year is how to make my ideas known to other people. I usually assume that other people are smarter than I am and I should really have more confidence in my thoughts. I have gone through the same courses and the have worked through a lot of similar problems, so my ideas and thoughts are valid. Even if I say something that isn't 100

What I Like / Dislike

I like the idea of the project in general, but I am somewhat disappointed with the implementation. I think the idea of making a code visualization tool is super cool, but the way we did it is very clunky and kind of unsafe, in a software sense. Our extension opens another application to display the UI. To me, that just feels really bad, but we really had no other option. I think with more time we would have pivoted and changed to an IDE that would allow for a more integrated way of making custom UIs. It's too bad we couldn't do that but at the same time, we made a really cool project in a short amount of time. I really like the grammar system of our extension. It's really nice that the User can define grammars and make the extension work on any number of project, but it's complicated. The grammar system is really specific and is not well documented at the moment and for users to use the grammar system they have to do a bunch of complicated regexes. So I guess I more like the idea of the grammar system and not so much the implementation.

If I Were The Client

If I were the client I would not be too satisfied with the product. I think it is a great proof of concept, but Scaffidi wanted more of a "useful tool." We did not know that that is what we were trying to make until we were already mostly done with the project, so we were bootstrapping a bunch of features for users that until then we were not even thinking about the users. So I'm not sure it all came together that well.

What to Work on Next year

We would need to fix all the small bugs and rework the displaying of the map. I think it would have to be a huge change to make the problems that I have with the extension currently go away. But with the work that has already been done it could be done in a school year. I think if a new team took it on they would be very lost in our code though.

26) Week 7: Sam Lichlyter: This Week

Most of this week consisted of preparing for expo and then actually being at expo. We put the last final touches on our pitch and gave it all we had.

Expo

Expo was great, we met a lot of people who were interested in our project and our tool. We got a few people who asked if we were looking for jobs after this term. Being recently accepted into grad school, I unfortunately had to say no to a few people but said I would reach out when I graduate. Eric I believe got a few peoples names.

A Retrospective

If I were to redo this project from the beginning, I would definitely put more time into the planning stages as well as more communication with our client. I think with those two combined we could have developed less and designed more and turned out a better product as well as relieving some frustration along the way. One of the biggest things I learned was how to work in a group effectively while still maintaining friendships. That I think is a valuable skill I will use in the workplace in the future. I am proud of the product we turned out, I am more impressed by the logic behind it than the visual aspect of it, just because it is so adaptable.

27) Week 7: Zach Schneider: What We Did This Week

The team's main focus was prepping for Expo this week. Our progress report due Monday included our Expo pitch, which summed up the general info we wanted to convey when people came up to us on Friday. After the progress report and a meeting about how we wanted to schedule our time at Expo, we felt we were ready to demo. We did last bits of testing to make sure

all features that would be demoed worked as intended. Expo itself went well. The first half of the day was slow; not many industry/academic individuals were present yet. After lunch time picked up a bit and we spoke to many company representatives. The only instance of being ”grilled” was by two engineers from Nvidia who asked us ”why would we use your tool instead of Intellisense?”. We responded that our tool was meant for a newer audience and would generally would be used on lighter platforms than Intellisense. Though this interaction was a bit intense, I feel we represented our project well overall. Most people who stopped by had generally positive remarks on our project, so I feel our project was well received.

Thoughts on the Year

This project has been extremely educational for me as far as showing the process of moving from design to complete implementation. Our primary issue early on was not nailing down exactly what we were trying to build. A lot of this was figured out as we built it, requiring a few redesigns in the middle of our work. My recommendation to myself for future projects, even if working with peers, it to have a definitive designer who makes the final design on all aspects of the project. As it was, the 4 members of our team each had a slightly different idea of what we were doing until we actually had to merge those features together. On this theme, I feel I have significantly improved my ability to work as part of a team on this project. Prior project only required that I do my own part, then add it to the rest of the team’s work at the end. This project required true coordination, a skill I believe is imperative to all software development.

If I were the client for our project, I would be happy with the design and core project that’s been developed. I’m not sure I would be happy with the product we presented at Expo, only because I know the particulars of what was presented: a small subset of the use cases our product claims to meet. Our product is on the verge of meeting so many more use cases and legitimate use, except that largely due to deadlines and burnout, we only developed what was absolutely necessary. I believe with perhaps as little as a few more weeks of full time development, our project could be very impressive. As it stands, it is a successful proof of concept only.

This project was a very beneficial experience. It taught me many lessons on software programming, both technical and team-oriented. It gave me a new perspective on what’s required in making a brand new piece of software. Though some of the classroom elements of this project may have been a bit contrived for the sake of a grade, the project itself was a practical experience. I know college can only have so many long term projects fit into it, but I feel a few more assignments of larger scale would be more beneficial than the week to week work we normally do.

28) Week 7: Eric Winkler: We’ve just finished expo but we still have several things to complete for this project. User testing still needs to be completed. To my knowledge we are waiting on IRB approval. I personally would like to do at least one more trial run. There are also a few very minor features I would like to implement that may help with testing. After trails are completed we may be asked to write a paper about the tool and the trails.

This project has been easily one of the more difficult things that I have attempted and I feel I’ve learned quite a few lessons from the work the team has done. Skill wise, I’ve become much more component with JavaScript and NodeJS. I’ve also managed to get a little better at git (conflicts are no longer have me resetting to head). I’ve also gained a lot of experience with design. When we had our midwinter refactor, we rebuilt the entire extension from the ground up and had to think about fitting all of the complex behaviors of the system together. I found that it is much easier to take a big problem and simplify it into smaller, manageable tasks. I’ve definitely learned a lot about abstracting.

This project has made me realize that I actually really enjoy the design/architecture part of software development and I think that is something I would really like to specialize in in my career.

Overall, this project was a success for me, personally. While it’s still a little early to gauge the success for our client, I have a good feeling about how our user trials will turn out based on the informal tests we have been running.

VII. Engineering Expo Poster

The poster displayed at the Engineering Expo is inserted in the following page.

VIII. Project Documentation

This section of the document covers the details of the projects implementation, and the ways that the user can expect to install and use the tool.

A. How It Works

The extension consists of three main pieces. The first, and most complicated, is the parser. The second is the visualization, and the third is the IDE Visual Studio Code. The user starts in IDE opening a code project. From the VSC IDE the user can start the extension, and then the parser gets to work on the users code. The parser is responsible for breaking the users code down into informational data structure that the visualizer can use to generate the map that the user can then navigate.

The parsing is done in several steps. It is implemented using a combination of functions built into VSC as well as some custom code using Node.js. The built in functions in VSC are mainly involve getting all the files in the user's current working directory. The custom code is the bulk of the parser. It is written in typescript and uses a number of Node.js modules, and it is responsible for checking the file for the things specified within the grammars that are supplied by the user. A grammar is a JSON object defined by the user and consists of regex patterns to define rules that the parser uses when creating the visualization file map. The parser then takes this grammar and generates a datastructure. The file map based on the defined links between files, or whatever the user specifies within the grammars. The map consist of nodes and sub-nodes that are representations of the files and their connections that the parser created. The map is shown on the visualization custom UI.

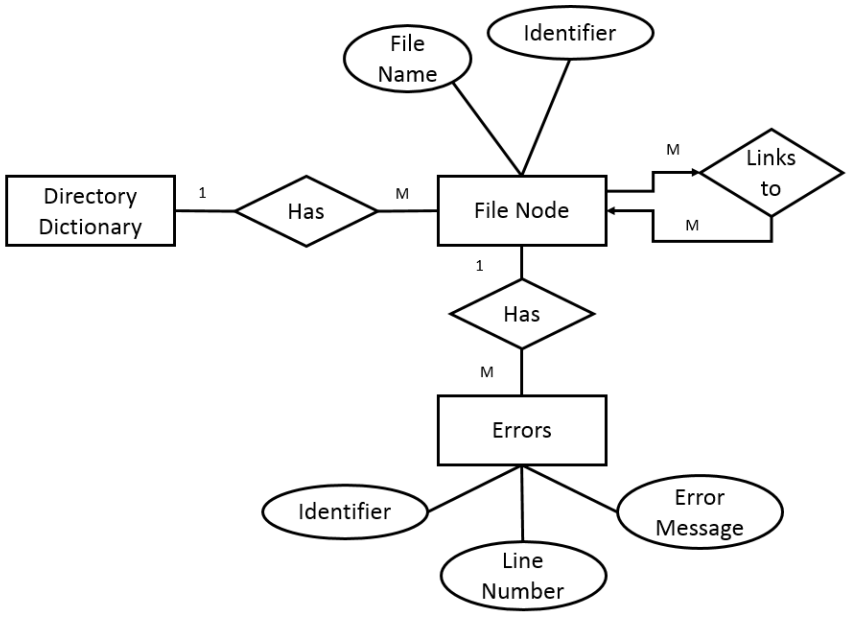


Figure 4. Control flow of the way the data structure is formed

On execution of a Visual Studio Code command, the Electron application will activate the parser generate the file map. The visualization is done in the separate Electron window that is opened. It has several things to display from the parser information. The main piece of information that the visualization displays is the file map and other information is the notification list. The file map is a graphic representation of the of the user's project solution. It appears as a hierarchical graph of interconnected

nodes where the nodes represent a file or directory in the user's project directory, and an edge represents some link (defined in the parser section) between the two files. The location of the nodes in the hierarchy will reflect its position in the project's directory. This web will feature nodes of different sizes and will allow the user to zoom and pan the view. The size of the node is based on the number of links to that object. A color corresponding to the type of file. Color values are based on the VSC color scheme for aesthetic reasons. The name of the node is retrieved from the file struct name field and is displayed as text beside the node. A red circle will appear on the node if there are notifications within the FileStruct for that node. In other words, if the size of the notification array within the FileStruct is not zero. The file map is rendered within its own div using the vis.js library 'Network' module. The Library by default includes the rendering, panning and zooming functionality.

The other piece of the projects custom UI is the notification list. The notification list will display all notifications currently in the project directory in the form of a vertical list. These notifications will be retrieved when the UI opens from the same data structure is being generated. These notifications will be retrieved in a per node fashion and will also be grouped in the error list in the same order. The notifications list will exist to the side of the file map in the same electron application screen. The list will allow the user to scroll when the number of errors result in the list exceeding the electron window height. When a notification in the list is hovered over, these errors highlight the corresponding node in the file map by changing the color value of said node. When a notification in the list is clicked, the extension will open the file in Visual Studio Code?s text editor and scroll to line where the notification location exists. The notification list is in its own div and has included scrolling functionality.

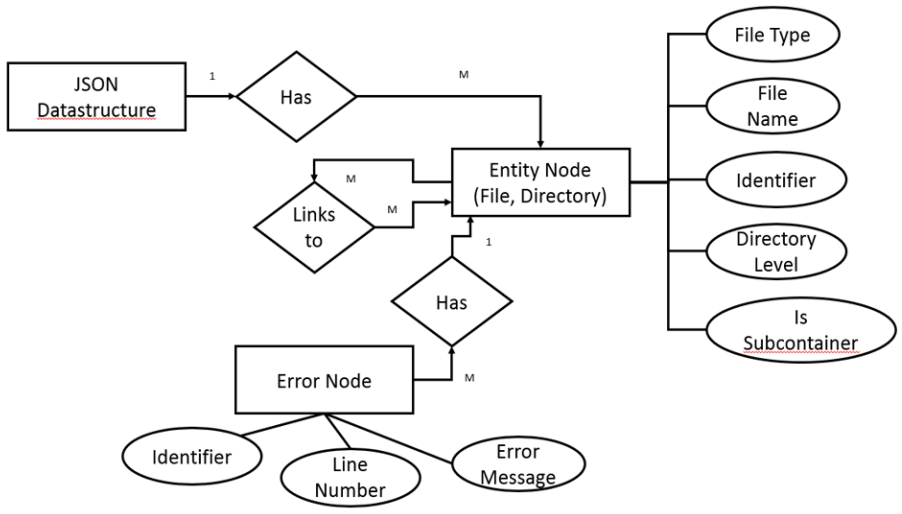


Figure 5. Information that is stored in the data structure and the way that is it interpreted by the custom UI and displayed

Data handling within the Postal Extension consists of three main entities or processes: the data structure, serialization of the data structure and the storage of the data structure in a JSON file. The data structure is a dictionary of file nodes, stored as JavaScript objects. These JavaScript objects come from the Parser parsing the currently loaded project for links and notifications. The data structure can be considered the live version of the project data, as its data structure is updated by the parser every time the project is parsed. The nodes changed in the data structure will then be serialized into JSON jQuery using the JSON.stringify

function built into JavaScript. This JSON will be saved to a file, which can further be read by the Electron UI functions for updating the project node and notifications visualization.

Now onto the IDE. There are two main interactions that happens the specific IDE events the opening of the custom extension UI and the interfacing network that allows the user to click on the custom UI in electron and have events happen in the VSC interface. The first of these events is opening the custom extension UI, this is done by creating a new process that runs the Electron window and passing the information that the parser has collected to that process. The interfacing network is opened and closed every time the user clicks a UI element within the custom UI. The network is a simple server client network with the server being the VSC IDE and the client being the Electron window. The VSC initializes itself as a server that will listen for the Electrons message when the user interacts with a UI Element. Once that happens the extension will navigate the users code window on VSC to the appropriate location that the user specified when they clicked on the custom UI.

B. Structure

The extension is developed with a Model View Controller design. The Model being a data structure that is created by the parser. The View being the custom UI displays a visualization of the data structure. The controller is the communication between the data structure and the UI.

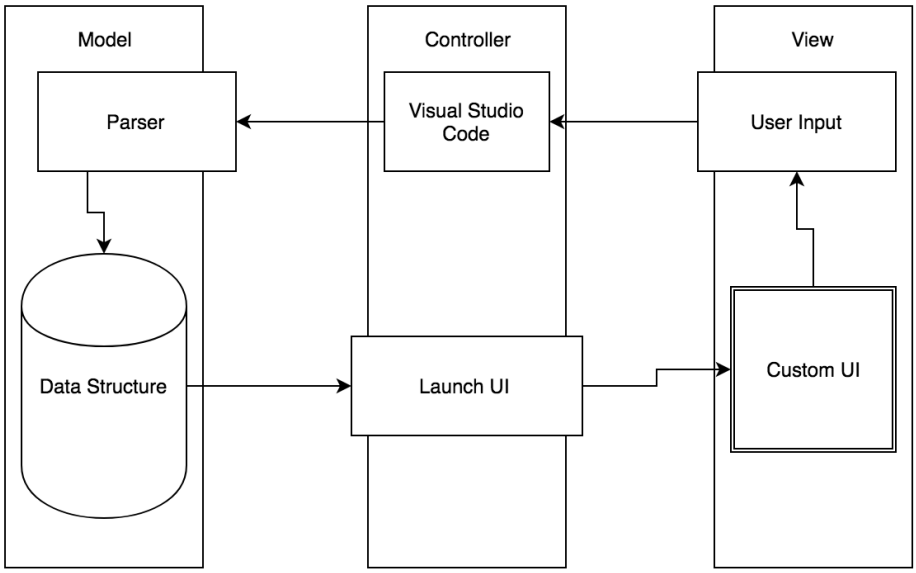


Figure 6. Diagram of the way the Model View Controller Design is implemented

C. Theory of Operation

Once installed and with the proper grammars made for the project that the user is working on the extension should function as follows. The user can then use the command `command + shift + p` to open the command dialogue. From there the user can input the `Postal` command. This operation will start the parsing of their files, and the creation of the custom UI that will open soon after inputting the command. Within the custom UI the user can navigate around the representation of their code using their mouse and zoom functionality. They will be able to view different code elements based on the grammars they have defined. The user is also able to view the notification list. The user can click on any of the nodes or any of the notifications and

they will be navigated back to the VSC window where a new tab will be opened and the area in the code that was specified by the element clicked will be displayed, and ready for the user to edit. The user is able to access and edit their custom grammars in the settings.json file that is part of VSC.

D. Installation

1) From Visual Studio Code Marketplace:

1) Within Visual Studio Code, click the extensions tab (the last one that looks like a block)
2) type ”postal” into the search bar
3) click the install button
4) Navigate to: `/.vscode/extensions/postal-team.postal-$version/`
5) Run the command `npm install`
6) Navigate to: `/.vscode/extensions/postal-team.postal-$version/lib/app`
7) Run the command `npm install`

2) From GitHub: Run the following commands:

1) `git clone https://github.com/slichlyter12/Postal.git`
2) `cd postal`
3) `npm install`
4) `cd lib/app`
5) `npm install`
6) `cd ../..`
7) `code .`
8) VSCode should now be open with Postal’s source code
9) Hit F5
10) Navigate to: `/.vscode/extensions/postal-team.postal-$version/`
11) Run the command `npm install`
12) Navigate to: `/.vscode/extensions/postal-team.postal-$version/lib/app`
13) Run the command `npm install`

E. Run Instructions

To run the extension once it is properly installed all on has to do is open VSC and open a project. From there the user uses the `Command + Shift + p` command and the VSC command dialogue will be open, and the user can then input the command `Postal` and the Postal Extension will start.

F. Prerequisites

To use the extension the user needs to have Visual Studio Code and be able to use the npm installer. To be able to run the npm install means that they have to have Node.js installed. They need this for the commands that are described in the Installation section. This is needed because the extension uses Electron to display the map that the extension builds, and Electron is too large to package and have put on the Visual Studio Code Extension Marketplace.

IX. Resources and Technologies

The most complex technologies the team had to learn was Node.js and vis.js

1) Node.js: For learning Node we primarily used the Electron reference guide available here: https://electron.atom.io/docs/ This was the most helpful resource in figuring out how to build an extension for Electron as well as basic node process information. The section on the Electron API was especially helpful.

The team also made use of a couple github projects for reference purposes. The biggest help by a wide margin was the VSCode Color Picker made by Anseki. https://github.com/anseki/vscode-color/ While the code was quite dense it was also essential in figuring out how to launch an Electron window from VSCode. It also gave the team some insight in how to better build the high-level structure of our extension. It is important to note that the team did not take any code from the project and only referred to it as a guide for using the correct Node and Electron functionality.

2) Vis.js: Vis.js is intuitive for doing basic things with graph visualizations. However, it can be very clunky when trying to implement more complex behaviors on top of its standard functionality. The main resources used for figuring out Vis.js related difficulties were the official documentation and examples webpages. http://visjs.org/docs/network/ http://visjs.org/network_examples.html The documentation itself was helpful for more specific information like finding function overloads but was not at all useful for understanding how to use Vis.js's functionality. Fortunately, Vis.js also provided a very comprehensive examples page with samples demonstrating the majority of supported functionality and function usage.

3) Additional Resources: We also made use of other technology specific websites and Stack Overflow for minor issues. Both Regex101 and Curious Concept's JSON formatter were very helpful tools. https://regex101.com/ https://jsonformatter.curiousconcept.com/ Additionally, we based the majority of our parser's architecture on the compiler we built in CS480. It was, in retrospect, surprising how useful that class was to completing this project.

X. Learning and Overall Experience

A. Sam Lichlyter

1) Technical Information: Since my main source of income over the past seven years or so is as a web developer, I am extremely familiar with JavaScript. However, I had never used TypeScript before, which is essentially Microsoft's version of JavaScript that has a few added functionalities, including types. I found this extremely helpful as one of my main complaints with JavaScript is that it is not typed. I also learned how to use Node.js which I had been meaning to do for quite some time. I had never heard of Electron before which was pretty useful to be able to essentially write a website that would be run as it's own application in it's own window. This project also turned out to be a compilers crash course. The parser we wrote was essentially a customizable compiler, so I learned a lot of the intricacies of how those work.

2) Non-Technical Information: I think one of the biggest things I learned was how much I appreciated some of the skills we were taught in the very beginning of our college career that I more or less disregarded when they were first introduced to me. The big one that stands out is paired-programming. I don't think this project would have been as successful as it was or completed in the time frame needed without paired-programming. Eric and I were able to successfully do this by him "directing" and me "driving." I was able to focus on the syntax and typing while Eric was able to basically tell me how everything was connected. If we had tried to do this on our own I don't think either of us would have been able to keep straight what had to be connected to what and we would have run into hours of debugging later down the road.

Another big thing I am now more appreciative of is designing before coding. They tell us this all throughout our college career, but usually our assignments aren't big enough to warrant this kind of design first approach. This project however definitely was.

3) Project Work: Piggy-backing a bit off of the last section, designing first is extremely important. Especially for projects of this magnitude and bigger like I expect a lot of professional jobs will have us work on. Our team didn't completely disregard designing the system before we started, we just didn't quite know what we needed, so it made it extremely difficult to design thoroughly.

4) Project Management: One of the biggest takeaways from this project about managing a team is communication. It is extremely important that everyone is on the same page, and everyone knows what the end goal is. This is something we ran into at least once. At one point we were all on one page and our client was on another, and at another point some of us were on one page and the others were on a different page. Keeping everything straight in a system like this is complicated, but everyone needs to know their roles and how they fit into the bigger picture of getting the whole team to where they want to be.

5) Working in Teams: This wasn't quite something I learned through this project but it was definitely exemplified in this project. And that was that everyone on the team has their own strengths and weaknesses. Some people are better at doing research into different technologies, others are better at conceptualizing large and complicated systems. These are things that are vital to each team and that team should use these abilities and talents to their full extent.

Another important aspect I learned about working in teams was that separating business from pleasure is very important. At the end of the day you all want to be friends but still have a really cool project to show off to people. You need to be able to separate personal feelings from what needs to get done, and then when the work is over, you need to not really talk about work until the appropriate time. For me at least, I found this very helpful to maintain the friendships I had with the people on my team.

6) What I Would Do Differently: If I would do it all over again, I would go through the advice I just listed above and implement it. I would put more effort into design first before implementation, make sure everyone knows their roles in the team and how they fit into the bigger picture, use the things we were taught in school for how they're meant to be used, etc.

B. Zach Schneider

1) Technical Information: The core development of this project involved the learning and utilization of many new technologies, languages, and programming principles for the team and myself. I had previously used VS Code, but had never developed an extension for it or any other Electron based code editing platform. Debugging an add-on and testing it on all relevant platforms is not only useful practice for future projects in that vain, but also gives me a great appreciation for the work other developers have done over the years in extending the tools I use and enjoy. This was my first foray into using TypeScript, a JavaScript super-set that includes types and class-based object oriented programming. I enjoyed working with TypeScript, and felt like it included many of the JavaScript programming strategies I'm familiar with while offering a more consistent and featured experience. TypeScript was primarily utilized in programming for our Node.js server within VS Code. I had used Node in one project previously, but our capstone project this year gave me a much more in depth view into how Node servers work and how to effectively program with them. Additionally, I learned and became proficient in the Latex document preparation system. Latex creates consistent, professional looking documents, and I used it to create most of the documentation and reports for this project.

2) Non-Technical Information: During this project I became more proficient in writing up formal documentation, particularly in adherence to IEEE standards. Learning how these kinds of documents are developed, updated and referenced will greatly aide me in all future programming jobs. I also believe my writing ability improved overall, as the many different reports required for this class stretched me in ways I hadn't previously been challenged to attempt. I learned how to merge the writing styles of others within a single paper in order to have a more cohesive submission that flows, instead of 4 separate sections pasted

into one document. Editing, while not re-writing is a skill of mine that has certainly been furthered while working on this project.

3) Project Work: Perhaps the most significant thing I learned about project design and implementation is how important it is for the designers and implementers to be on the same page. Even in our case where each of our team members served both roles, we had quite a few problems early on in not understanding what design or project feature the others were trying to convey to us. Sometimes we mistakenly wrote code in an entirely different train of thought than other sections, leading to serious conflicts in our project's structure and function. Ultimately, our solution was increased communication and more time put into the planning stages of each component's section. Along these lines, our team struggled to even conceptualize what we wanted from our project as a whole and if our concepts would actually work. In a few cases, we wrote a design or specification for a feature, then realized while creating that feature that the original envisionment would not work out. What I can take away from this is, again, how important the planning and design stages of projects are to their success. Experience in working on projects like ours will help when starting on new projects in the future.

4) Project Management: One of the issues our project faced was running out of time at deadlines. Even if they were self-established deadlines, we couldn't strike a balance of having a deadline be significant while also flexible enough to accommodate series design problems that we occasionally encountered. My time management in these contexts was certainly tested in trying to allow for enough time to accomplish what was needed while also not setting deadlines too far into the future. Additionally, I learned the importance of differentiating when a feature or task should be done versus when it needs to be done. Setting these standards for myself and communicating them to my team effectively contributed positively to my knowledge of project management.

5) Working in Teams: Team work can often be difficult - personality clashes, differences in work ethic, lack of communication - these are all fairly expected when working on team projects. While our team was generally able to learn and work through many of these standard issues, a unique circumstance we possessed was that we are all friends outside of school. Some of us had heard warnings to not work with friends on projects, but we didn't really pay them heed. Throughout this project, it has been exceedingly difficult to draw the line between our work relationships and our social relationships. Often we would finish up work on the project then promptly spend the rest of the weekend hanging out. While this led to an easy dynamic early on, our relationships became strained when serious problems occurred, such as individuals not meeting deadlines, not communicating or forgetting about meetings. Though our project worked out in the end, this experience is not one that I'd like to undertake with friends again. In the future I will aim to let friends be friends and work colleagues be work colleagues.

6) What I Would Do Differently: With the aforementioned lessons being learned, if I were to do this project again, I would do a few things differently. First, I would not choose to work with close friends on an extended and stressful project such as this. I would make sure to spend a more significant amount of time in the early stages really nailing down what the project's purpose, goals, design and features will be. Communication about these plans would be made with the team and our client much more frequently at the beginning. Coding would begin in early December instead of January, as the extra month of working through bugs would make a large difference in final quality and lessen deadline stress along the way. Finally, I would begin utilizing user testing far earlier, even before features are complete in order to get feedback and save the team some of the re-writing that occurred in the later phases of the project. Despite these changes I would make, I feel the majority of the project went well and I'm proud of the result.

C. Cramer Smith

1) Technical Information: This was the largest project that I have ever made, and we wrote it in JavaScript and TypeScript. I learned a lot about JavaScript and Node.js. I learned the usefulness of code organization. I always knew that having well organized

code was good, but I had never had a long term large project that really needed it. It was great after we had done a major part of the development adding smaller features to the tool was much easier because we already had all these small pieces well put together and these parts could easily be put together to make something larger. I also learned about thenables. Thenables are a very annoying and incredibly flexible way of working with asynchronous function calls. I had never used TypeScipt and for any function that was not working as I expected is because it was probably running asynchronously. To fix it I quickly learned that a thenable was needed. I also learned about using a client server relationship to communicate between processes. To get the user events between the custom UI with the Electron window and the VSC editor we needed to communicate between two different processes. I always though that processes were very closed off entities, and they are unless you tell the processes to communicate to one another. To do that communication I set up the client server.

2) Non-Technical Information: I learned some self confidence. Usually in team project I do not try to be the idea man. I have always assumed that others are smarter than me, but this project I learned that I have some good ideas. So I tried to make myself more heard during this project.

3) Project Work: It is a lot of work to get a big project working, and a lot of work can be saved by a good initial design. We had an okay initial design. The core ideas of that initial design are still there, but some of that design did not work as expected. We had to redo a lot of work because of the initial design was not sufficient. But even if we had had a perfect design it would still have been a lot of work. The amount of hours that we put in to get the tool working, was more than I thought it would be.

4) Project Management: Communication is extremely important. I knew that before. It was really evident though when we initially delegated the separation of work between the four of us, and we all started developing without talking too much to one another. Then by the time we were all supposed to be done with our respected parts we came together and none of the parts worked together. We should have been talking with each other more in that crucial development phase. That being said I think I would have liked some more structure with the way we all managed the project. It was nice when we nominated the team leader to handle all the document deadlines. It was nice to know that if I was unsure of what I needed to do for X document I could ask Zach. That was the only real structure that we had though.

5) Working in Teams: It is hard to not have emotions rise. There were times when it was very tense in out Slack chat. I am still not sure how to remedy this problem. I think it would be different in a professional environment. I delegation is good, but only with really good communication. There needs to be a good understanding of the expectations of the team.

6) What I Would Do Differently: Besides the differences that I stated in the above sections I wouldn't be in a team with my friends. I think this hurt my friendships with these people, and I was probably less motivated because I knew my friends would be more forgiving.

D. Eric Winkler

1) Technical Information: From a technical perspective, I've learned about or become more proficient in:

- JavaScript and TypeScript
- Node.js
- Electron
- Vis.js
- Git and Github
- Other Web-technologies (HTML, CSS)

- Software Design
- Asynchronous design
- Object oriented Programming
- Debugging
- Compilers
- Using APIs in general
- Reading Documentation

2) Non-Technical Information: The theme to the non-technical lessons I've learned while working on this project is that I have a new appreciation for a ton of things I learned here but hadn't used in a practical way before.

To start with, I now love paired programming. Building the Parser and back-end of this project was potentially the most complicated thing I have attempted during my undergraduate career. It was almost necessary to have Sam writing code while I focused on keeping track of what exactly we were doing and what we still had to do. Writing most of the code together probably saved us tens of hours.

I also learned to love classes. I've never had to use classes on the basis of trying to abstract complex problems but being able to describe things in terms classes and forget about lower level details was very helpful.

Finally, I learned to appreciate the importance of designing before coding. I'm not sure I've ever completed a homework assignment by first designing it before writing the code. Everything had been, relatively speaking, simple to implement. However, without an assignment guide or anyone to hold our hand through the process, we were forced to actually draw out the components of the system and really think about why we were implementing things the way we were. I think the most valuable piece of non-technical Information I gained from this project would be that I really like to do design related work. I may attempt to pursue a career path where I can do more architecture and systems design than programming, provided that exists.

3) Project Work: The most valuable lesson about project work I learned is that if a group plans on splitting the work into separate modules and assigning one or more modules exclusively per person, said modules should still be designed together as a group. The team has mentioned this problem before in previous documents and it should be mentioned again how important it is to have well defined interfaces. What I've never brought up is that those interfaces imply that the design and functionality of the two modules are already clearly defined and understood by the team. That's the more important lesson to note. It doesn't matter at all what the exact specifications of the interfaces are if there is missing or overlapping functionality in those modules. If I had to come away with only one important lesson from this project it would be this: design thoroughly with all members of the team present.

4) Project Management: From a project management perspective, I've learned the importance of setting one's own deadlines and setting them early. Around the middle of winter term, the team concluded that we were starting to fall behind of where we wanted to be in terms of implementation progress. We had made the decision to get a working build of our extension on the VSCode store by the next coming Monday. Nothing was specifically due for the class but we had placed this artificial deadline on ourselves because of concern that something might cause problems if we had planned to be finished by the end of the term instead. Because of the how tight our deadline was, we forced ourselves to work close to 36 hours within a three-day period. During that crunch we discovered serious problems with our design that would have caused the project to be a failure if we had waited much longer. My opinion is that deadline may have saved the project.

5) Working in Teams: I was fortunate to work with an excellent team. Everyone was motivated and completed their assignments on time. We had no major problems and communication was consistent. One lesson I learned is that it helps quite a bit to have at least one person who can act as a leader. This person doesn't necessarily have to delegate jobs or roles, but having an individual to check on team member's progress and keep a general level of organization is beneficial.

6) What I Would Do Differently: A problem the team had that we don't reflect on much is that we had a bit of miscommunication on project priorities with our client. We had spent so much time on our back end figuring out how to get our parser working consistently that we had neglected the front end quite a bit. In a meeting our client was not happy with our progress, though a substantial amount had been made since our last check in. We had failed to realize that as a tool, it was only as good as it was usable, i.e. its front end. I regret not frequently communicating with our client and making him a larger part of the process.

XI. Appendix 1

A. Code Samples

1) Example Grammar: This is the default grammar that parses HTML and PHP files for divs and links.

```
"Postal.grammars": [
      {
         "title": "div",
         "type": "tagged",
         "filetypes": [
            "html",
            "php"
         ],
         "options": {
            "tagStart": "<div",
            "namedOption": "id=\"(.+?)\"",
            "tagEnd": ">",
            "closingTag": "</div>",
            "nodeColor": "blue"
         }
      },
      {
         "title": "href link",
         "type": "link",
         "filetypes": [
            "html",
            "php"
         ],
         "options": {
            "link": "href=[\"](.+?)[\"]",
            "nodeColor": "blue"
         }
      }, ...
]
```

2) Recursive Get All Links: This function grabs all the links from the data structure of a specified file struct and it's children.

```
// Recursive function to get all links from this and children
function getAllLinksFromFileStructRecursive(FileStructID) {
   var links = [];

   // check parent
   if (DFS[FileStructID].links.length > 0) {
      for (var i = 0; i < DFS[FileStructID].links.length; i++) {
         var link = DFS[FileStructID].links[i];
         links.push(link);
      }
   }

   // check children
   if (DFS[FileStructID].subContainers.length > 0) {
      var childLinks = [];
      for (var i = 0; i < DFS[FileStructID].subContainers.length; i++) {
         var childFileStructID = DFS[DFS[FileStructID].subContainers[i].toFileStructid].id;
         childLinks = getAllLinksFromFileStructRecursive(childFileStructID);

         // push what we found to parents link list
         for (var j = 0; j < childLinks.length; j++) {
            links.push(childLinks[j]);
         }

      }
   }

   return links;
}
```

XII. Appendix 2

A. *Images*

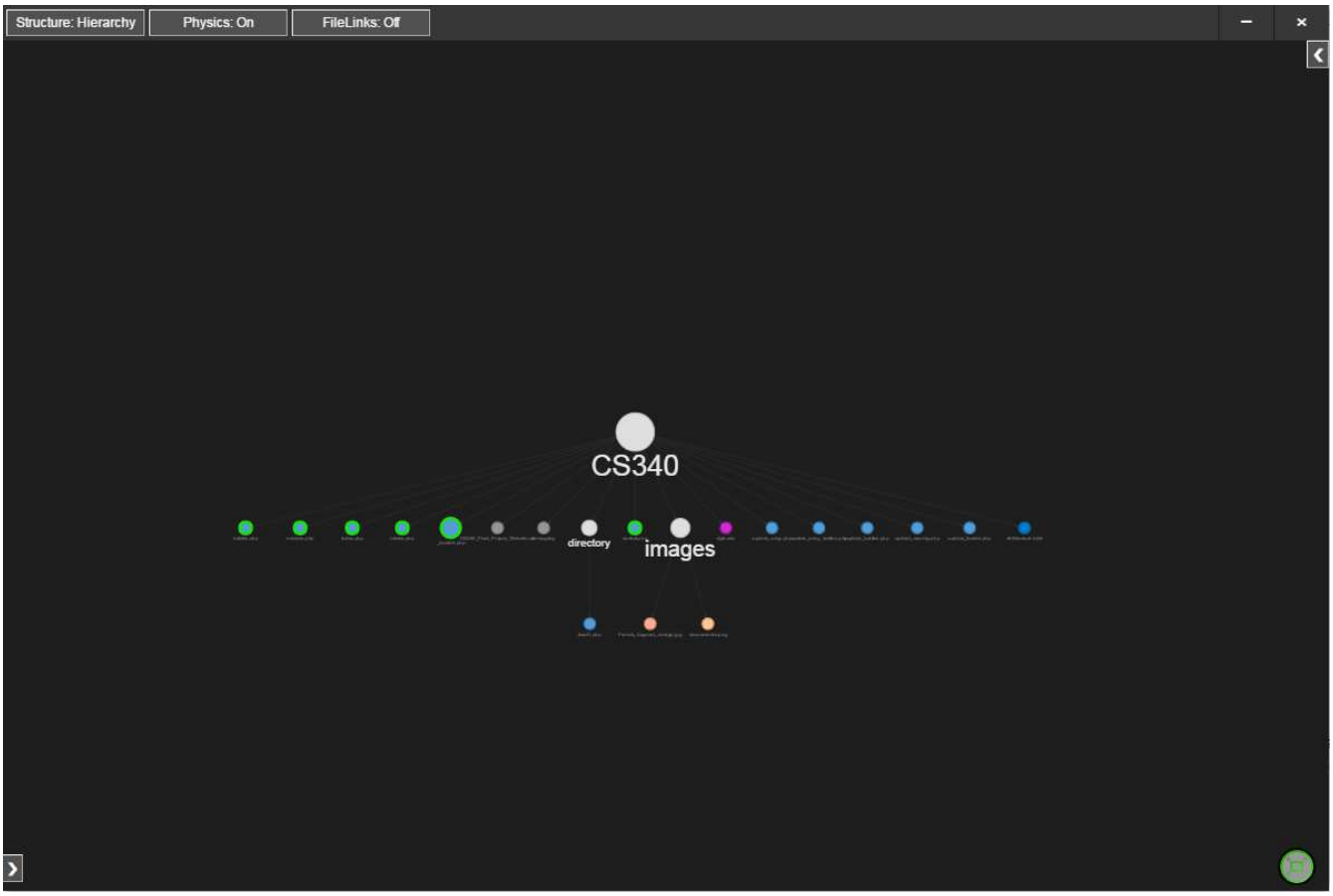


Figure 7. Visualization Interface

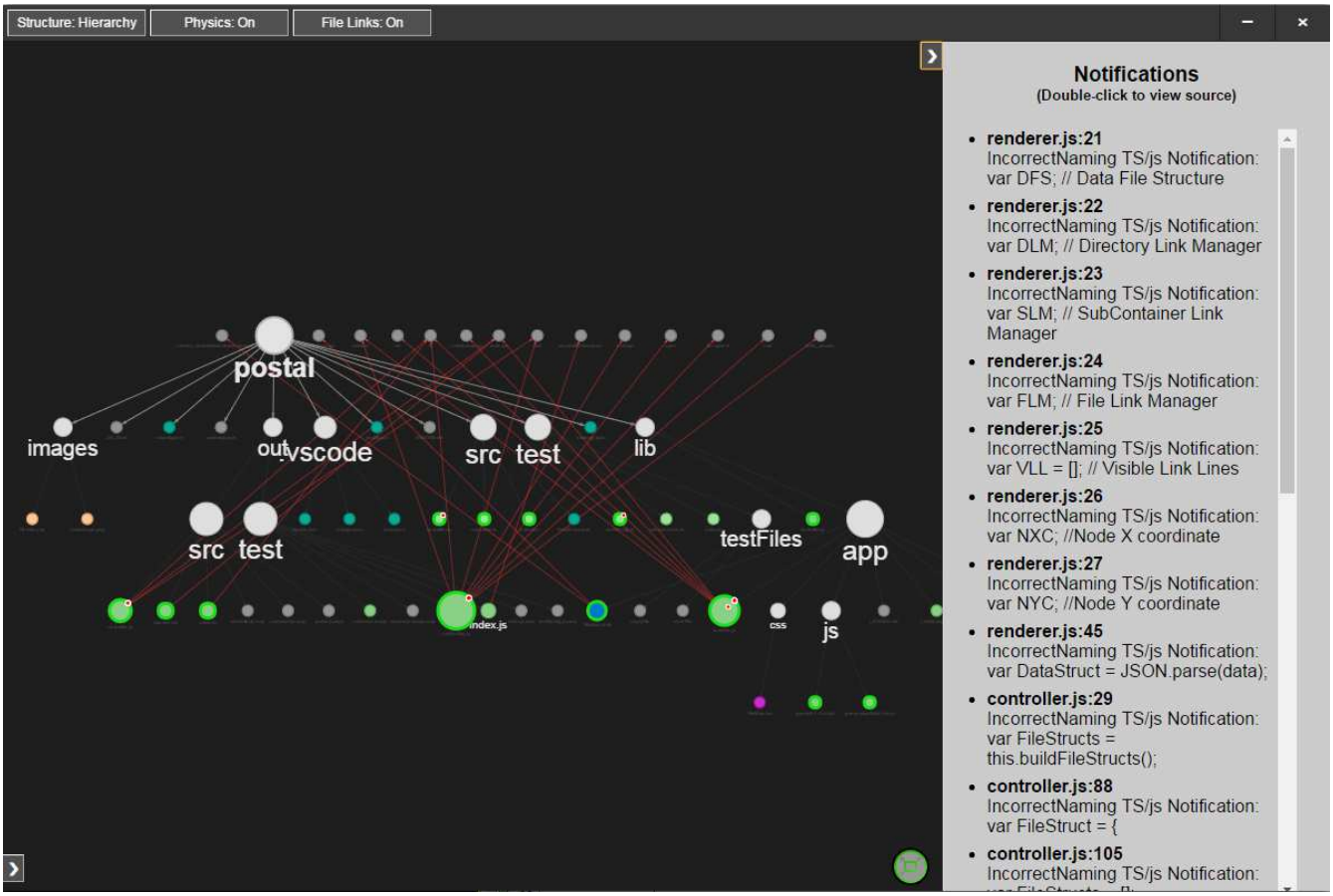


Figure 8. Notification Interface