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 1, 2016

CONTENTS

I Overview 4
I-A Scope . . . 4
I-B Purpose . . . 4
I-C Intended Audience . . . 4
I-D Conformance . . . 4

II Definitions 4

III Conceptual Model for Software Design Descriptions 4
III-A Software Design in Context . . . 4
III-B Software Design Descriptions Within the Life Cycle . . . 5
III-B1 Influences on SDD Preparation . . . 5
III-B2 Influences on Software Life Cycle Products . . . 5
III-B3 Design Verification and Design Role in Validation . . . 5

IV Design Description Information Content 5
IV-A Introduction . . . 5
IV-B SDD identification . . . 5
IV-C Design stakeholders and their concerns . . . 5
IV-D Design views . . . 5
IV-E Design viewpoints . . . 5
IV-F Design Overlays . . . 5
IV-G Design Rationale . . . 5
IV-H Design Languages . . . 5

V Composition Viewpoint 5
V-A Design Concerns . . . 6
V-B Design Elements . . . 6
V-B1 Extension . . . 6
V-B2 Parser . . . 6
V-B3 File Map and Error List UI . . . 6
V-B4 Data Handling . . . 6

VI Logical Viewpoint 6
VI-A Design Concerns . . . 6
VI-B Design Elements . . . 6
VI-C File Map and Error List UI . . . 6

VI-D Design Concerns 7
VI-E Design Elements 7
VI-E1 FileMap 7
VI-E2 Error List 7

VII Interaction Viewpoints 8
VII-1 Design Concerns 8
VII-2 Design Elements 8

VIII Information Viewpoint 8
VIII-1 Design Concerns 8
VIII-2 Design Elements 8

IX Interface Viewpoints 9
IX-A Design Concerns 9
IX-B Design Elements 9
IX-C UI 9
IX-C1 Exposes 9
IX-C2 Requires 10

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 includes the team implementing the extension, their client, and their supervisors. The teams supervisors being the people grading the project on the implementation of the designs described within this document.

D. Conformance

The document conforms to the IEEE Std 1016-2009.

II. Definitions

III. Conceptual Model for Software Design Descriptions

The 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 roles . The model will be the data structure that we will use to represent the parsed files. The view will be the user interface and the integrated development environment. The IDE and the user interface that the extension creates will be the view and act dependent of each other. The control will be the event handlers that are

A. Software Design in Context

The extension is supposed to help novice web developer with organizing and create cleaner HTML and CSS code. To accomplish this the design of the extension it will contain a UI, a number of parsers for the different languages. The parsers and the extensions will need to communicate between each other and VSC. This communication will be facilitated by a data structure that will also allow for a

B. Software Design Descriptions Within the Life Cycle

1) Influences on SDD Preparation:

2) Influences on Software Life Cycle Products:

3) Design Verification and Design Role in Validation:

IV. Design Description Information Content

A. Introduction

B. SDD identification

C. Design stakeholders and their concerns

D. Design views

E. 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

F. Design Overlays

G. 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.

H. Design Languages

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

V. 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

B. Design Elements

1) Extension: Type: system Description:

2) Parser: Type: component Description:

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 Datastructure. The Datastructure is a series of JavaScript objects and values populated by the Parser, containing details on the project currently loaded into VSC. TODO: ADD DETAILS ON DATA Structure The Datastructure will be serialized into structured JSON objects by the JSON.stringify function and saved to a file. This file will be the direct source from which the UI obtains its data.

VI. 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

B. Design Elements

1) Parser:

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. The user interface will then be implemented using JavaScript, HTML and CSS.

C. Design Concerns

D. Design Elements

1) 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.

2) 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 Jquerey Advanced news Ticker.

datahandling (datstructure, saving)

VII. Interaction Viewpoints

The Interaction viewpoint defines strategies for interaction among entities, regarding why, where, how, and at what level actions occur.

1) Design Concerns: The concern that come with using these events is that we might not be the correct events that the extension needs to be listening to. It could be that doing the parse on every save will slow down the development process too much. If that were the case there would have a specific command that the user would need to input every time that they would want to use the Postal Extension. This scenario would greatly decrease the usability of the extension by adding an unnecessary step to the process.

2) Design Elements: ¡¡¡¡¡¡¡ HEAD Parse Files on savefile step 1 .... step 2 ... open UI vs code command ======= ——Parse Files on savebox

There are several actions that occur within VSCode that will initiate the extension parsing and interpreting process. These events will be specific events that will have specific listeners. Each event will trigger a specific type of the same process. These events are when the user starts the use of the extension, when the user explicitly saves one or all the files, and when the user closes the application. When the user starts the extension the extension will the first initial parsing and create the data structures that will serve as a reference for the next continuation of the extension. This first parsing will be set in motion by the built-in activate function. After the initialization whenever the user saves the files the extension is planned to parse the files and get the necessary information to parse once again. The extension will only continue on the explicit 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. This should trigger the parser and continue the extension process, but it will continue it before it has actually saved the documents. This will allow for possible formatting of the user's code before it is saved. Then 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.

——open UI vs code command

When the a parsing of the files happens it is important for the extension to know what has change in the user current work space.

¿¿¿¿¿¿¿ 757893940a66d4d9e78a6b80fbfbda17b8d8d62f

——Open error object

VIII. Information Viewpoint

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

1) Design Concerns:

2) Design Elements: Dictionary

FileNodes fileName file type ect.

Errors line number error string.

IX. 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.

A. Design Concerns

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

B. Design Elements

IDE Exposes Open GUI Save File Requires

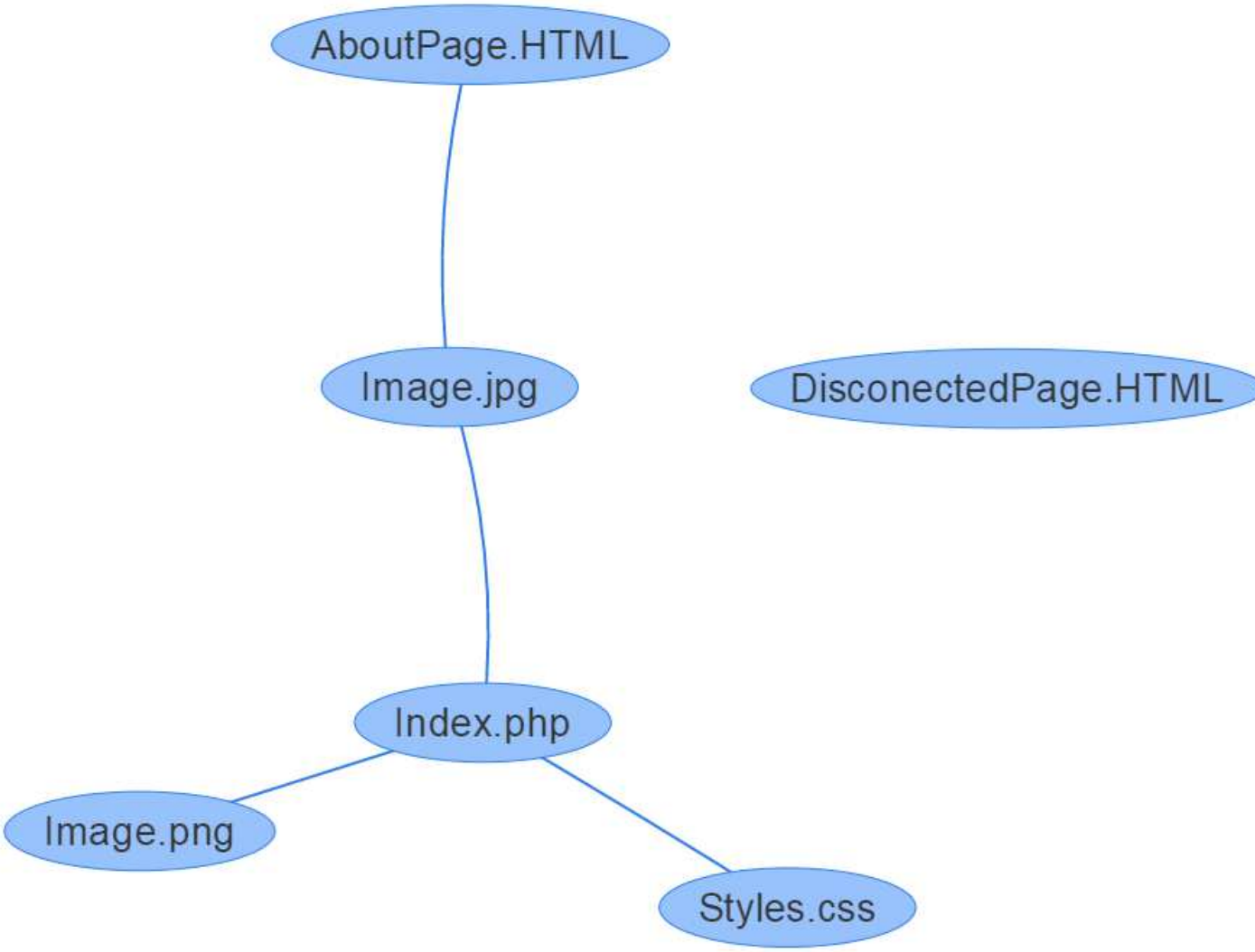
Parsers Exposes Parse Requires Save File (IDE) Get Data (DataStructure)

DataStructure Exposes Get Data Requires Load Data (Files) Parse (Parsers)

C. UI

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 (DataStructure)
- Open GUI (IDE)

Files Exposes Load Data Save Data Requires Get Data (DataStructure)