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

Technology Review and Implementation Plan

Team Postal

Cramer Smith, Sam Lichlyter, Eric Winkler, Zach Schneider

November 15, 2016

Abstract

CONTENTS

I. INTRODUCTION

II. SAM LICHLYTER

A. Parser Class

1) PEG.js: PEG.js is a parser generator. Its sole function is to generate a parser which is saved as a JavaScript Object with which we can interact. It also generates a small API for us to use. This would reduce a ton of work we would have to do as far as writing our own parser. The problem with this is that we would only have a simple API that we could interact with. Theoretically we could modify the API it generated for us but then we would have to learn how this parser was generated.[1]

2) UglifyJS: Uglify JS looks as though it could also generate a JavaScript Parser for us. It has the added bonus of being able to generate a source map of our source code which could be helpful for our own file map. This although has all the same problems that PEG.js does.[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. For these reasons I think this is the route we will take.

B. Data Structure

1) Technology 1:

2) Technology 2:

3) Technology 3:

C. Piece 3

1) Technology 1:

2) Technology 2:

3) Technology 3:

III. ZACH SCHNEIDER

A. Data Structure Serialization and Storage

The text editor extension created for this project function 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. JSON.stringify, a mongoDB database, and the serialize-javascript npm package will be considered for the serialization of our data structure.

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. [3]

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. [4]

3) MongoDB: MongoDB is a free and open-source database solution that uses JSON-like documents for storing data. It is a NoSQL database, meaning it differs from typical SQL-based relational schema systems. 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. MongoDB is supported on numerous cloud and local platforms, scaling from small local uses to large server applications. The two main drawbacks for using MongoDB is that it has a large overhead to install (hundreds of megabytes) and none of our team members have used a NoSQL database before. For this unfamiliarity and space requirement, our team has opted not to use MongoDB to serialize our extension data. [5]

B. 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. [6]

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. [7] 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. [3]

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. [8]

C. Piece 3

1) Technology 1:

2) Technology 2:

3) Technology 3:

IV. CRAMER SMITH

A. The Best Base Integrate Developement Enviroment

It is nesecary to look all the possible integrated developement environments for the postal project to see what would be the best fit for this specific extension, both developement wise and release wise. To this purpose this review will examine three IDEs that could all be used as the base of the extension that is plannned 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. [9] 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 that are examined 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 advantageous to have that ability. 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 extension manager in brackets is not user friendly, it is basically a list of extensions and a search bar. With the target audience we are hoping to reach we think that the extensions need to be more readily available and easier to understand.

2) Atom: Atom 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. [10] 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.

3) Visual Studio Code: Visual Studio Code is an editor by Microsoft, and it was the editor that was initially proposed as the base of the extension. [11] This initial idea to use 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 this was 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 plans on doing extensively.

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. [12] 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 without the use of PHP.

2) Built In Events: The Extension could fire events that are built in to Visual Studio Code and have handlers set up to listen to those events .

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 receives 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 interact with. [12]

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. The second reason being that the HTML preview would be extremely limited, tedious and impractical.

C. Languages Extensions Written in

With all the possible IDEs the extension can be written in HTML, CSS, and JavaScript, but the bulk of the code is JavaScript. With Visual Studio Code there is the option to use TypeScript. This portion will look at JavaScript and TypeScript looking at each language's pros and cons.

1) JavaScript: With all the possible IDEs the extension can be written in JavaScript and corresponding JSON files. JavaSctipt was created in 10 days in 1995, is a well known and well documented language. [13] It can be very confusing at times as it is an asynchronous language and not compiled inline.

2) TypeScript: TypeScript is a language created by Microsoft with the purpose of being 'JavaScript that scales.' [14] 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. [14] 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.

3) Decision:

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. Piece 3

1) Technology 1:

2) Technology 2:

3) Technology 3:

VI. CONCLUSION

REFERENCES

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

[2] Mihai Bazon. Uglifyjs. Lisperator.net. [Online]. Available: http://lisperator.net/uglifyjs/

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

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

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

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

[7] 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

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

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

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

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

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

[13] 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

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