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

Winter Midterm Update

Team Postal — Group #38

Cramer Smith, Sam Lichlyter, Eric Winkler, Zach Schneider

CONTENTS

I. Project Purpose and Goals

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 obtained faster. Information Foraging Theory (IFT) 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 for IFT and increase developer efficiency.

The Postal extension is being designed to allow developers to more quickly search through and better visualize their projects. This extension will also help developers create clearer and cleaner code structure by offering reminders and suggestions about the best coding practices in the programming language they are currently using. Any major errors or incompatibilities within the project or its files will be reported to the developer as well.

II. Current Status: Fall Term

As of the end of Fall term, the Postal team has completed all documents and preliminary requirements needed to move forward with the creation of the Postal extension. Initial prototypes for both UI elements and file parsing components of the extension exist and are beginning to be further developed. The team aims to have a fully working prototype by the end of Winter break that is able to parse and display any web development project. Current and prior progress indicates that the Postal team is on track to meet this self and client imposed deadline. Below are current descriptions and functionalities provided by the major components within the Postal extension.

A. Parsers

The current implementation of the extension has one parser which parses HTML. This parser is implemented in Perl using the Simple TokeParser which looks for specific tags and a specific attribute from that tag. For example, this HTML parser specifically looks for the anchor (`a`) tag and the hyperlink reference (`href`) attribute and returns those values which are the links to either outside websites or internal pages.

B. User Interface

The user interface is currently running off of dummy, static data. It opens up in a new window using a program called Electron as opposed to a web browser to cut down on the overhead associated with all the complexities of the web browser. It shows each web page as a node and each link from one page to another as an edge in a graph. See Figure 1.

C. Data Layer

The data layer is comprised of a data structure and data access methods. The data structure is defined below in the ERD (Figure 2).

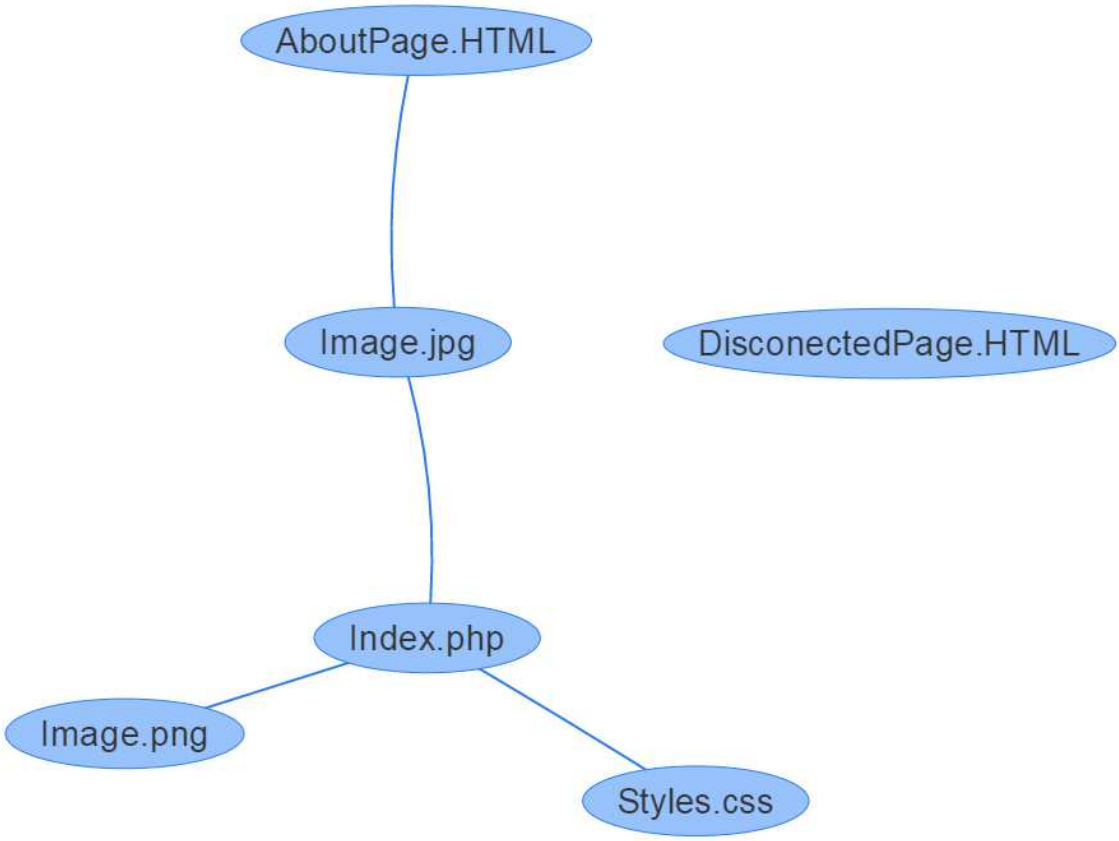

Figure 1. User Interface Example

III. Problems and Solutions: Fall Term

A. Pre-Week 3

The beginning of our project was back in August when Sam told the rest of the team about an opportunity that he had to work with Prof. Chirs Scaffidi on a senior design project. The members of the team all jumped on the opportunity to work together for a professor all of us had had previously and throughly enjoyed. Prof. Scaffidi tasked the team with coming up with project proposals. The only requirements for the project is that it had to be a tool that used information foraging theory to do anything. The first problem was that none of the team knew what IFT was so the team took a week to brainstorm seperately and research IFT and come up with project ideas. After that week the team met and decided on two ideas to give to the client. Scaffidi chose a code base organization tool that would be an extension for Visual Studio Code.

B. Week 3

After the first couple weeks of class while other teams were just meeting the Postal team was able to start meeting and thinking about functionality as well as begin to work on the problem statement. The problem statement was the first document of the project and it was a challenge to create an official document and learn how to use the

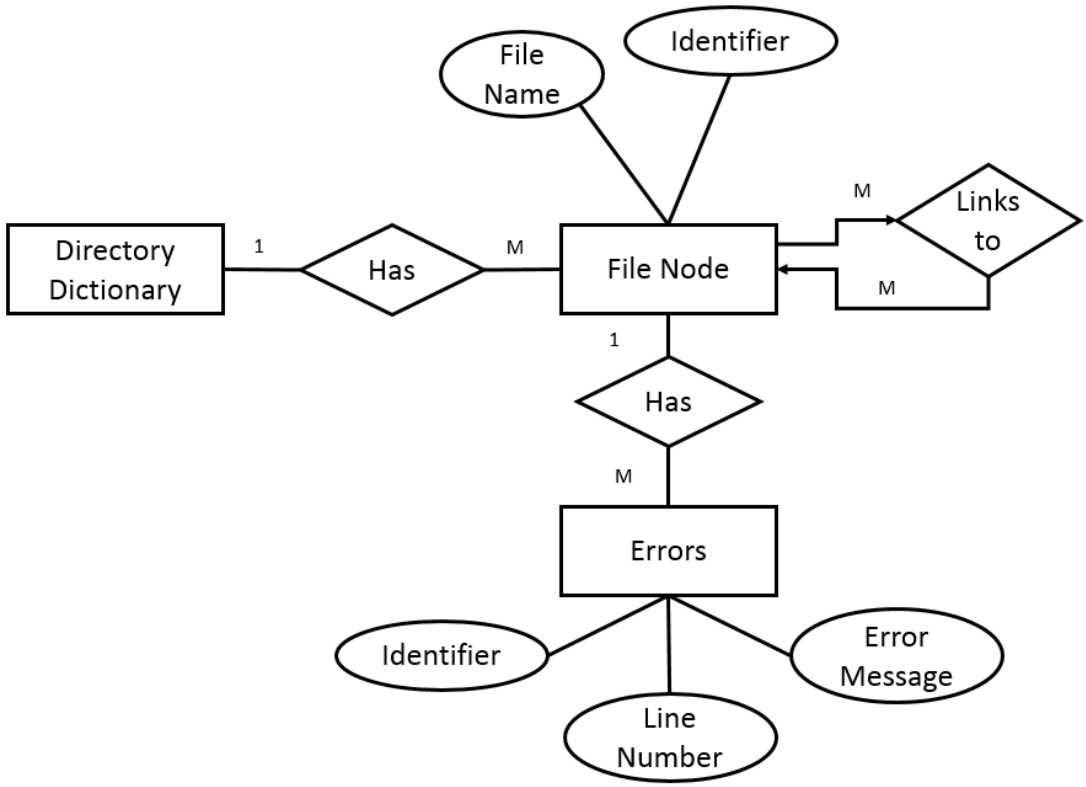

Figure 2. Data Structure ERD

IEEE standard. This problem was not really solved. The team ended up creating what they thought was a good document, and there was feedback on the document that came later in the quarter helped tremendously for the next draft of the document.

C. Week 4

During the fourth week, our client had challenged the team with accelerating the development process past the class. In response to this request the team started developing basic ”hello world” extensions with very limited to no functionality. These programs served as a learning experience for the team as we develop more complex extensions these examples gave us a framework of understanding of what the code of an extension looks like. These small programs were good but not up to par with our clients expectations.

D. Week 5

In week five the team revised the problem statement. This task was fairly easy as the team had received feedback on the first draft and therefore had a knowledge of what was expected of them. This week we also started work on the Requirements Document. The requirements document was much more daunting than the problem statement

as it was more technical, and made the team think more the project as a piece of software an not just a problem to be solved. The team wrote a first draft of the requirements document and made a plan to finish it the next week.

E. Week 6

With the completion of the requirements document the team had some solid ideas as to what the project was going to look like. This was a big step forward for the project as the team had only a fuzzy idea as to what the project really looked like at this moment. Now the team knew that it was going to be a web development tool instead of a general code organizer. This was great for the development of the project, but this narrowing of scope was not communicated to the client. The team thought that they had communicated this in a meeting but apparently either our client did not understand or the idea was not stated clearly. This caused problems later in the process.

F. Week 7

This week we thought was a big step for the team. By this time there was a working prototype of the user interface, and a simple parser for HTML. The only thing was that these programs were not actually a Visual Studio Code extension but separately compiled programs. The team needed to figure out how to combine the program UI with Visual Studio Code.

G. Week 8

The eighth week the team finished the tech document. The tech document made the team look at alternatives for parts that each member had decided to focus on. This was eye opening as we researched alternatives to pieces of the project that the team members had never thought of before.

H. Week 9

This week was Thanksgiving week. With the extra time Sam and Eric were able to further the prototype and make some small improvements. These small improvements still had not added the UI to the extension but the team thought that it would be good enough to show to the client.

I. Week 10

Week ten was a big week as the design document was due that week. This document proved to be the most confusing as the team had a particularly difficult time figuring out the IEEE standard of the software design document. The team was able to struggle through the design document and create what the team thought was good document. The team also was able to show to the client what we had done with the prototype so far. The client was happy with the prototype, but encouraged the team to reconsider the reduction in scope that had occured previously. This will be factored into the work going forward.

IV. Prototype Code Samples

A. FileMap UI

B. HTML Parser

```
my $anchor_parser = HTML::TokeParser::Simple->new(handle => *DATA);
while (my $anchor = $anchor_parser->get_tag('a')) {
   next unless defined(my $href = $anchor->get_attr('href'));
   say $href;
}
```

V. Fall Term Retrospective

Topic	Positives	Deltas	Actions
Communication with client	• Flexible in terms of project requirements. • Provided the team with a mock up application architecture. • Easy to work with.	• Frequency and clarity of communication need to be improved.	• Solidify team member responsibilities (Sam is in charge of emailing the client). • The team as a whole will need to make sure Sam is aware of things that need to be communicated with the client.
Team Effectiveness	• Members have flexible schedules and are easy to sit down with. • Members all do a good job of quickly replying to communications. • Members have a positive attitude.	• As a team, assignments do not get much planning and are usually done last minute. • There is no clear team leader.	• Assign a pseudo team leader with the responsibility of tracking/ scheduling tasks and work assignments.
Implementation	• The team has become much more capable at developing with the VSC IDE. • The Team has already built a couple of working prototypes for the client.	• The Gantt chart is currently out of date and needs updating. • Some technical implementation obstacles have yet to be addressed (npm auto installing).	• Update Gantt chart to better reflect the current progress and set new goal deadlines for winter term.
Design	• The current design meets requirements and team members are aware of their design related responsibilities.	• IFT Design Pattern tie-ins are not 100% clear and as a major requirement of the project, they need to be. • Parse needs to be redesigned to better match our scope. See Week 10 notes.	• The team will need to meet to re-design the parser component of the extension and review the selected IFT design patterns.
Documentation	• All assigned documentation is currently in a workable state.	• Team members feel that the documentation is typically rushed and not a descriptive as it should be. • Team members have a difficult time interpreting the IEEE standards and this may be a partial cause to the above.	• A team member will take a leadership position to schedule work tasks which will ensure that the team has sufficient time to complete the documentation assignments to a higher standard.

VI. Current Status: Winter Midterm Update

A. Overall

Currently a working version of the extension is being distributed on Visual Studio Code Extension Marketplace. This is the beta release of our project showcasing the major functionality of our extension. The main feature of the extension is the code base visualization. This was a significant milestone for the development team, but there are still features left to be added to the extension. The next big step in the development process is to add the process bridge to allow for user interaction within the custom UI to navigate to specific areas in VSCode. After that we plan on adding defined errors for certain web languages. We are aiming to meet these goals by the end to winter term. Once the development process is completed the focus will change to doing formal user testing.

B. Parser

As of week seven, the parser is nearly complete. It will need to be updated to support same line searching and comments. For example, when users have multiple HTML divs defined on the same line, the parser currently incorrectly parses this. To solve this problem we will need to change the parser to look at files character by character. The current version of the parser looks line by line for specific strings specified by the user in a user-editable file, grammars.json. The advantage of this was so that users could specify what they needed and the extension wouldn't have to have built-in support for a multitude of languages. First the parser looks at all the users files and directories open in the project directory they have open in VSCode. These files then get passed one-by-one to a parser class which tokenizes the file based on the grammars the user defined. The tokens are then used to create subcontainers and links which are built into the data structure for each file. Lastly the parser writes the data structure to the data layer by use of a controller.

C. User Interface

The users main interaction with the extension is via an Electron window that displays their project as a graph of nodes and edges, see figure 3. The user launches this app by running "Postal" from the command palette within VSCode. This command calls the parser and then calls the UI to open once the parser has finished writing the data structure to the data layer. Currently the UI represents the users project as a hierarchy view, this will later become user editable between the default hierarchy view and another "web" view. The user can interact with their project by double clicking on a node that has a subcontainer, for example a .html file that has divs in it. This will expand the subcontainers underneath it. Now the .html file they double clicked that had divs declared in it, those divs will now be displayed underneath it's parent. This logic continues recursively, so that nested divs will be shown underneath their respective parents. Another feature of these nodes is that if they have an "id" specified (this is more specifically targeted towards divs by default, but again, this is a user editable parameter), these ids or names will show up instead of the default "div" (also specified in the grammars file).

Figure 3. Current User Interface

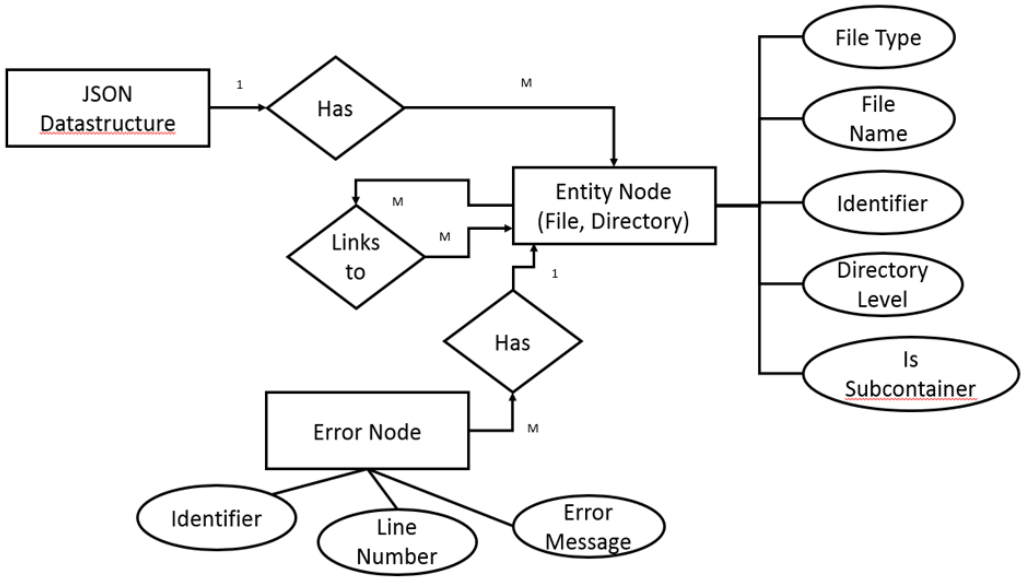

Figure 4. Updated Data Structure

D. Data Layer

The data layer has been redesigned to simply be a storage source for the most recently parsed project data. It is still stored in a single JSON file, which is written to on each parse. The actual arrangement and writing of the data occurs in a controller file which received file, node and link data from the parser. The controller then takes this data, appends it in the correct order to a JavaScript object which holds all nodes and subsequently writes to the JSON file.

VII. Problems and Solutions: Winter Midterm

A. Winter Break

Winter break most of the members of the team got together and discussed the work that would need to be done over the winter quarter. Once the schedule was put in place the team assigned work and continued the development of the extension. The main focus of this work was trying to get the different pieces to function together properly.

B. Week 1

Week one there was a meeting scheduled with our client. With this hard deadline so close there was an influx of work as this first meeting was when the team was planning to have the mostly functional extension. We where still having some problems with launching the custom UI from within VSCode, and we were still working with data that was not generated from project files. The reason we were not using generated data was because we were still having trouble parsing out the information to create the logical links between the files within the projects that the extension was trying to visualize. We met with Chris Scaffidi and showed him the visual structure that our extension could create with the non-generated data, and he seemed really happy with the results. Afterward he gave us a deadline for February to have a working alpha and even some of our friends testing our extension.

C. Week 2

With the success from week one we lulled into a much slower work flow. The team did not get a lot of major work done this week only some minor updates to the parsing the files, and more work done on the custom UI. These changes were very minor and most of them were later deprecated.

D. Week 3

In week three we worked on more minor changes updating the UI, and very basic error handling. Once again these changes were later deprecated.

E. Week 4

Week four we started to figure out what we needed to update in the design and tech documents as well as set up a simple OneNote engineering notebook to document our changes to these documents. With the notebook started we started working on changing the documents. At first we only focused on the grammatical errors.

F. Week 5

Week five we worked on redoing our tech and design document. A majority of the parts of the two documents stayed the same. The main differences in the documents were the addition of the process bridge, no more use of the eventEmitter, and the more advanced details about the parser.

G. Week 6

The weekend before week six was a big jump for the development of the extension. The whole team got together and did a code review over the different parts of the extension. The code review led to the each members of the team realizing that the separate parts were functioning differently than the other developers had thought. This confusion of the different parts was the cause of the parts not functioning together correctly. After this realization we decided to rework the main parts of the extension. We started with the parser. Sitting down and creating a more concrete design that would break up the files more and get more pertinent data. Using ideas that they had learned from studying compilers Sam and Eric were able to create a system for breaking up the files into more manageable token based data objects. These token objects made for a much easier time making the links between the files that we were having so much trouble with all quarter. With the new token and links system we also had to redo UI because it was used to the old data model. In total the whole rework took all weekend and a good part of Monday. The result being we now have a much better idea of the direction in which the extension should go design wise, and the code is much cleaner. We were even able to publish the extension on to the Visual Studio Code Extension Marketplace. Publishing the extension did introduced a small problem that we expected in the beginning of fall term, and that was that the node packages might not auto install. We still are not sure how to resolve this on user's machines that do not have node installed.

VIII. Prototype Code Samples

A. Example Grammar

This is the default grammar that parses HTML and PHP files for divs and links.

```
{
   "id" : 0,
   "title" : "html",
   "filetypes" : ["html", "php"],
   "rules" : [{
         "title": "div",
         "type": "tagged",
         "options" : {
            "tagStart": "<div",
            "namedOption" : "id=\"(.+?)\"",
            "tagEnd": ">",
            "closingTag": "</div>",
            "nodeColor": "blue"
         }
      }, {
         "title": "href link",
         "type" : "link",
         "options" : {
            "link": "href=[\"](.+?)[\"]",
            "nodeColor": "blue"
         }
      }, {
         "title": "includes link",
         "type": "link",
         "options": {
            "link": "include=[\"](.+?)[\"]",
            "nodeColor": "blue"
         }
      }, {
         "title": "body",
         "type": "tagged",
         "options" : {
            "tagStart": "<body",
            "tagEnd": ">",
            "closingTag": "</body>",
            "nodeColor": "blue"
         }
      }
   ]
}
```

B. *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;
}
```

IX. Winter Midterm Retrospective

Topic	Positives	Deltas	Actions
Communication with client	• Sam has taken primary responsibility with the client and adequately communicated with him. • Client has been satisfied with project process at all meetings this term. • Feature deadlines and goals have been clearly set and agreed upon.	• Expectations concerning user testing to be made clear.	• The team will meet with our client to solidify user testing details for the coming months. • The team will continue to update our client with our progress.
Team Effectiveness	• Team communication has been relatively constant and quick throughout the term. • The team has been willing to put in extra hours when deadlines require them. • Tasks have been clearly delegated to each member.	• Accountability for getting assigned tasks done in a timely manner needs to be improved.	• The person who delegates tasks will work and communicate more with members when those tasks aren't being completed.
Implementation	• A working Beta version of the extension is now available on the VS Code extension marketplace. • A team code review was recently conducted to help each member understand all parts of the extension.	• NPM auto-installing once the extension is downloaded is a holdover problem from last term. • Error parsing and UI tooltips have yet to be implemented.	• The team will research how other extension utilize NPM. • The team will continue to create and assign tasks to reach remaining feature goals.
Design	• The team updated the design document to reflect the current status and goals of the project. • The parser and data layer have been redesigned to support future features. • Eric and Sam have reworked the UI to be more intuitive.	• Parser may need to be modified to parse character-by-character to identify specific elements within files better.	• The team will redesign and modify our parser functionality when we arrive at the feature that requires it.
Documentation	• All documents have been updated to reflect current project status. • Zach has taken responsibility for setting up and submitting documentation.	• Team members should reference the documents more frequently so that code doesn't differ from design as much.	• Documents will be updated more frequently (biweekly) as changes are made instead of waiting until term deadlines.