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

Spring Midterm Update

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. 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.

Our project (code-named ”Postal”) is an extension for Visual Studio Code. It 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

A. Project Status

As of the end of week 6 of Spring term, all requirements of our project have been met, the extension itself is fully functional and most bugs in the codebase have be either resolved or negated. Our extension is available for download and installation from the Microsoft Visual Studio Marketplace and is labeled as our 1.0 release. Previous alpha and beta versions had been available on the Marketplace but lacked some of the final functionality or had significant bugs present. Project Postal works and has been tested on all 3 major OS platforms, Windows 7/10, MacOS and Ubuntu specifically. The extension can parse and visualize most major programming languages, having been tested primarily on C, C++, HTML, CSS, JavaScript, TypeScript and PHP. Default grammars (a series of regular expressions that search for language-specific keywords or tags) that allow the parser to process and visualize most of these languages have been included with the extension installation. The program is highly extensible, allowing any user to add to existing language grammars or create new languages grammars themselves. To this effect, the team intends to demonstrate Project Postal at the Engineering Expo with grammars not included in the default installation, so as to show how grammars can be designed toward specific codebases.

Two deficiencies of note remain in our project that were not able to be resolved by the week 5 code freeze: a separate Node.js installation and minor visualization bugs. At this time, the Postal Extension requires Node to be independently installed on the user’s machine in order to work. The reason for this additional installation requirement is that our display and window system, Electron, is too large of a package to be bundled in with a Visual Studio Code extension. As such, the user has to use Node’s ’npm install’ command to acquire Electron separately. The other remaining bug of note in our project is an occasional inconsistency in how a given codebase is visualized. Postal uses the Vis.js package to create a network of interactive visual nodes that represent files in the user’s codebase. Occasionally, Vis will generate a network that has some file nodes set to display in a location far from the rest of the network. This visualization bug can typically be overcome by either redrawing the network (a button

available to the user) or closing and reopening the Electron window. The team was not able to determine the cause of this problem, but we feel it does not significantly detract from the overall functionality of the program.

B. Testing Status

According to our agreement with our client, Prof. Chris Scaffidi, the team will conduct user testing to demonstrate whether the IFT principles with which the program was designed are effective. The team has both written up a series of tasks for a set of users to perform with and without the Postal Extension, as well as a post-testing questionnaire that will gauge the whether the users felt the extension helped them achieve the tasks faster or more effectively. These tasks and the questionnaire were both approved by our client and have been submitted to the university's Institutional Review Board (IRB) for human subject research approval. As of the end of week 6 of Spring term, the team is still waiting for response from the IRB. When approval is received, the team and our client will set up a selection of users to test our program. The actual testing procedures have been defined in the User Testing section later in this document.

III. Remaining Work

A. Postal Extension

While Functionally complete, these are several minor features the team is planing on implementing before user trails begin. The first being a simple reset button built into the extension UI. This will reset the state of the vis.js map to its initial positions to be used in the event of buggy behaviour or as a quick way to collapse all sub nodes. Additionally we would like to implement functionality that starts all files within a visualized directory collapsed into a directory. Currently, directory nodes are not collapsible but really should be. One final change to make in the extension is a fix involving file links. Occasionally when a user rapidly expands nodes and enables/disable the file link button, the file links may become broken. While we haven't been able to find the exact cause of this bug, we plan on fixing it in time for testing.

B. User Testing

A large portion of work that is still in progress is user testing. We are currently tuning our test questions in preparation for human trials within the next few weeks. Several members of the group have done trial runs with our current test questions to get feedback before the actual tests begin. So far the trials have been useful in finding problems with the exams and we plan on continuing to tune until we get approval from our client to begin human trials. After human trials, we will potentially be responsible for writing up our findings in a publishable format which may take a significant amount of time.

IV. Problems and Solutions: Fall

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. Chris 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 thoroughly 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. The first problem was that none of the team knew what IFT was so the team took a week to brainstorm separately 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. Dr. 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 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 that 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 functionality. These programs served as a learning experience for the team. As we developed 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 client’s 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 us. 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 about the project as a piece of software and 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. The team decided 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 either our client did not understand or the idea was not stated clearly. This caused problems later in the year.

F. Week 7

This week 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 wrote the technology review document. The tech review made the team look at alternatives for software components that each member had decided to focus on. This was eye opening, as we researched alternatives that the team members had never considered 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 document. The team was able to power through and create a quality piece of documentation. 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 occurred previously. This would be factored into the work going forward.

V. Problems and Solutions: Winter

During Winter break, Sam, Eric and Zach were able to meet up and begin dividing work amongst themselves since they are all from the same hometown. We began coding some of the 'pieces' of the parser and data structures, as well as a basic UI canvas on which we could place test elements. Cramer independently worked to have our code function as a extension that would run in Visual Studio Code.

A. Week 1

We planned a meeting with Prof. Scaffidi at the end of that week, and as such, planned to have some kind of prototype to show him. Putting some of the pieces of our parser, UI and extension code together took much longer than we anticipated so our prototype wasn't exactly functional. Despite this, Prof. Scaffidi was happy with our progress thus far and set a few deadlines he wanted us to meet by the end of the next two months (February and March).

B. Week 2

The team had a meeting this week to evaluate where we and the project stood. We assigned further work in each of our respective roles. Eric got the UI to display an example codebase as nodes and links for the first time.

C. Week 3

The week was fairly slow for development. The team worked individual on their components and we all met after our meeting with Vee to make sure we were on track. Zach spent some time on the code that creates file nodes and links so that Eric's UI would correctly represent them visually.

D. Week 4

This week saw more progress as Eric got the UI to launch with VS Code automatically for the first time. Sam and Zach worked on the parser and data structure to make sure it output adequate file node information for the UI. Cramer got code highlighting working within VS Code, which will be useful for a future feature.

E. Week 5

We wrapped up file node creation from the parser in the data structure. The team worked separately on their respective roles.

F. Week 6

This was mostly an administrative week. The team worked together to update our various documentation to our current project design and decisions. Zach set up the OneNote engineering notebook for our midterm progress report.

G. Week 7

The team conducted a code review during the weekend, bringing us all to a better understanding of each others' code components. Unfortunately, it was at this time that we realized that the parser and data structure would need a slight redesign to accommodate the UI elements that we wanted. The rest of the week and the weekend were spent rewriting this code to get the project to a fully working prototype state, an alpha stage. Once this was reached, we published our extension on the VS Code extension marketplace for the first time.

H. Week 8

We met with Prof. Scaffidi this week, and while he was happy with our general progress, he still believed the usefulness of the UI was not where it needed to be. This prompted us to move from further work on core functionality and to focus on usability of the UI. We convened that weekend as well and spent much time coding to that end.

I. Week 9

Each team member worked individually, adding minor components to the UI and some functional elements of our extension. We added buttons that allow the user to configure the shape of the file node network in the UI and began adding the slide open window for a notification menu.

J. Week 10

The team worked this week to complete the preliminary version of our Expo poster. Additionally, we continued adding and improving UI elements and set goals for the beginning of Spring term.

VI. Problems and Solutions: Spring

A. Week 1

This week a meeting with our client did not go well. His feelings were that the UI was not refined enough and he was disappointed that we had not been using the tool ourselves. We immediately shifted focus onto our UI and wrote grammars for TypeScript, since that is what most of Postal is written in.

B. Week 2

We doubled down on our efforts to refine our UI and spent most of this week to that end. Another meeting with Prof. Scaffidi was scheduled for the end of this week. He was much happier with our additional changes and our use of the tool on itself. Because of this, the team moved forward in preparation for user testing.

C. Week 3

A formal user testing proposal was approved by Prof. Scaffidi and sent off to the university's IRB for final approval. When we receive that approval, we will move forward with user testing. The team spent the rest of the week completing our Expo poster board.

D. Week 4

Changes made to the team's expo poster board were not up to specification and needed to be reverted. Additionally, the team has had trouble getting the extension package to be uploaded to the Postal Market Place. Cramer is currently trying to find the issue and has put in a ticket with the developers for help.

E. Week 5

The team has made some changes to the user trials sent to our client but we have been having a little bit of trouble finding qualified pre-test subjects. To fix this, the team began asking more class mates and other senior project teams for help.

F. Week 6

This week the team began doing trial runs in preparation for human trials. This led to the discovery of several minor problems with our test cases. First, both of our tests were being completed with about five minutes left on the timer. Because of that, we are not able to really determine if using our tool significantly increases the speed that a user can find information. To fix this, the team will increase the length of the test. Additionally, we discovered that we had not written all of the grammars necessary to run one of the test cases. the team has resolved this by writing those grammars. Finally, the team also noticed that a question asked about finding the number of TODOs in a directory that did not have any, resulting in some confusion. The team made adjustments to fix this.

VII. User Testing

A. Testing Description

The user test procedure, in its current state, contains two sets of tasks that go with two specific software projects. One is a C++ project and the other is a HTML and PHP project. Both of these projects are about the same size and are of similar complexity. The test subjects will go through the projects completing the tasks that we have designed to measure the effectiveness of the use of IFT.

1) Experiment Procedure:

1) In a computer Lab with 15 computers set up the extension on the 15 computers.
2) On the 15 computers load up the 2 testing projects.
3) On the 15 computers set up a document with the question that they will be answering.
4) Have the subjects each sit at a different computer and give them a 5 minute tutorial on the Postal Extension and how to use it.
5) Direct the subjects to the questions that they are answering and the project that they are answering the questions about
6) Give the users ten minutes to complete the set of questions.
7) At the end of the ten minutes direct the users to the next set of questions and the other project.
8) Give them another 10 minutes to answer the question without the use of the extension.
9) Once the ten minutes are over they will be given a short survey.
10) The subjects will have 5 minutes to answer the survey.
11) Once they are done, ask that group to leave and bring in the other 15 subject and repeat the same procedure but switch the project that the subjects use the extension on.
12) Collect results.

The tasks are as follows:

For the **C++ project**

1) Locate the line at which void RenderingDirector::rdRender() is defined.
2) Find the number of functions in the snake.cpp file.
3) There is a function that is defined in Snake.cpp, but is never used. Name that function.

4) Find all the TODO comments in the program.
5) Identify the return type of the Snake::Turn function.
6) Find the number of times the Food class referenced in the gameplay-director.cpp file.
7) Find the number of .h files included in main.cpp.
8) Locate the file in which the Punish() function is defined.

For the **HTML and PHP project**

1) index.php is the homepage for this web application. Over time, various features and pages were added and removed. This resulted in some files that exist within the directory that have been deprecated (they are not used within the index page or any of the pages linked to index.php). Locate all the files that are deprecated. (select all that apply check list)
2) Find the number of divs in header.php
3) Some website applications make use of several external references like jQuery and Bootstrap. Find all external references used in the project directory. (External references are links to outside resources that are not included within the project directory, and are hosted on external servers)
4) What file and line number is site-menu declared in?
5) How many references are there to stylesheet.css in the project?
6) How many images are linked in the entire project?

B. Measuring Results

We will measure how long it takes the tester to complete the tasks as well as how well they answer the questions as a percentage. For example if a user were to find all the TODOs they would receive a 100% for that question. If they missed one of the ten they would get a 90%. If the subject failed to find any of the TODOs they would receive a 0% for that problem.

We will also do a post test questionnaire. The questionnaire will pose the subject with a number of questions that the subject can answer on a spectrum from strongly agree to strongly disagree.

The questionnaire is as follows:

1) I could effectively complete the tasks and scenarios using the Postal tools.
2) I was able to complete the tasks and scenarios quickly using the Postal tools.
3) The organization of information and interface elements of the Postal tools within Visual Studio Code was clear.
4) It was easy to use this tool.
5) I would use Visual Studio Code again.
6) I could think of multiple scenarios for using for the Postal tool.
7) I would use the Postal tools within Visual Studio Code again.
8) Seeing a project visually was useful.
9) I felt the file link feature[1] allowed me to complete the tasks more quickly.
10) I felt the file link feature[1] allowed me to answer the questions with more confidence.

11) I felt the expand feature[2] allowed me to complete the task more quickly.

12) I felt the expand feature[2] allowed me to answer the questions with more confidence.

13) I felt the notification feature[3] allowed me to complete the task more quickly.

14) I felt the notification feature[3] allowed me to answer the questions with more confidence.

[1] File link feature: Internal references to other files displayed as red lines within the node network.

[2] Expand feature: A right-click on any file node expanded the node network to include all sub-nodes (classes, functions, divs, etc.) of the indicated node.

[3] Notification feature: Notifications (regarding malformed code, incorrect styling, etc.) were indicated by a red circle attached to the associated files/nodes and described in the notification window on the right-hand side of the screen.

The questionnaire will help us to measure how the user feels about the usefulness and general feel of the extension. The more that they strongly agree with the statements that will mean that the users enjoy and feel that the extension is useful.

VIII. IMAGES

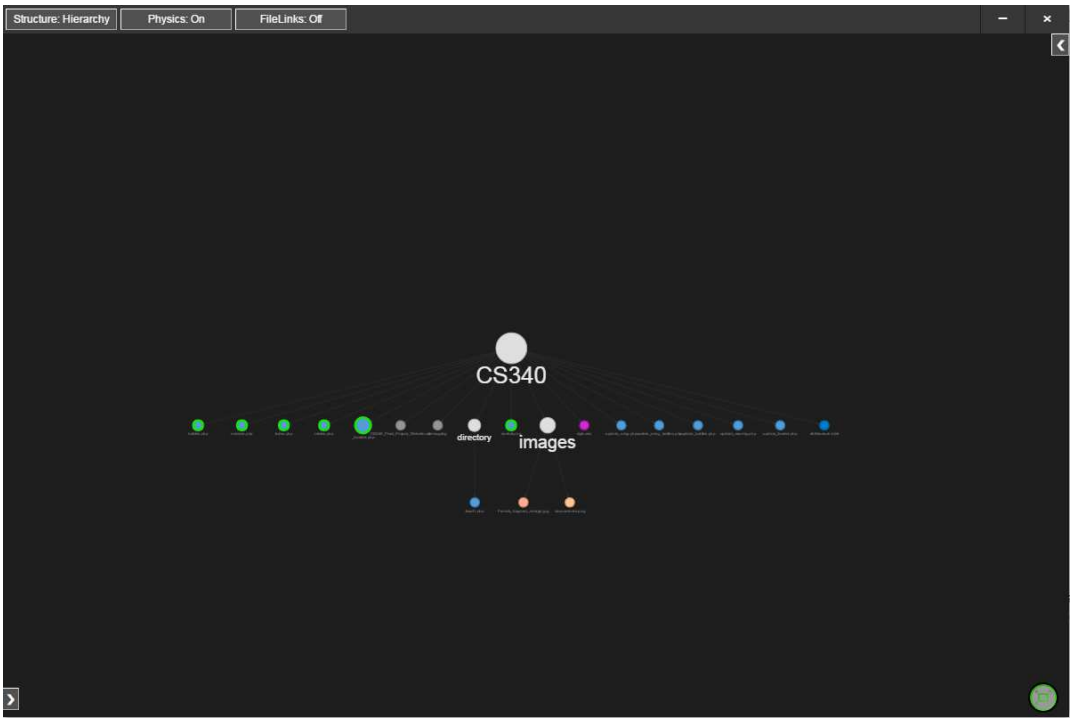

Figure 1. Visualization Interface

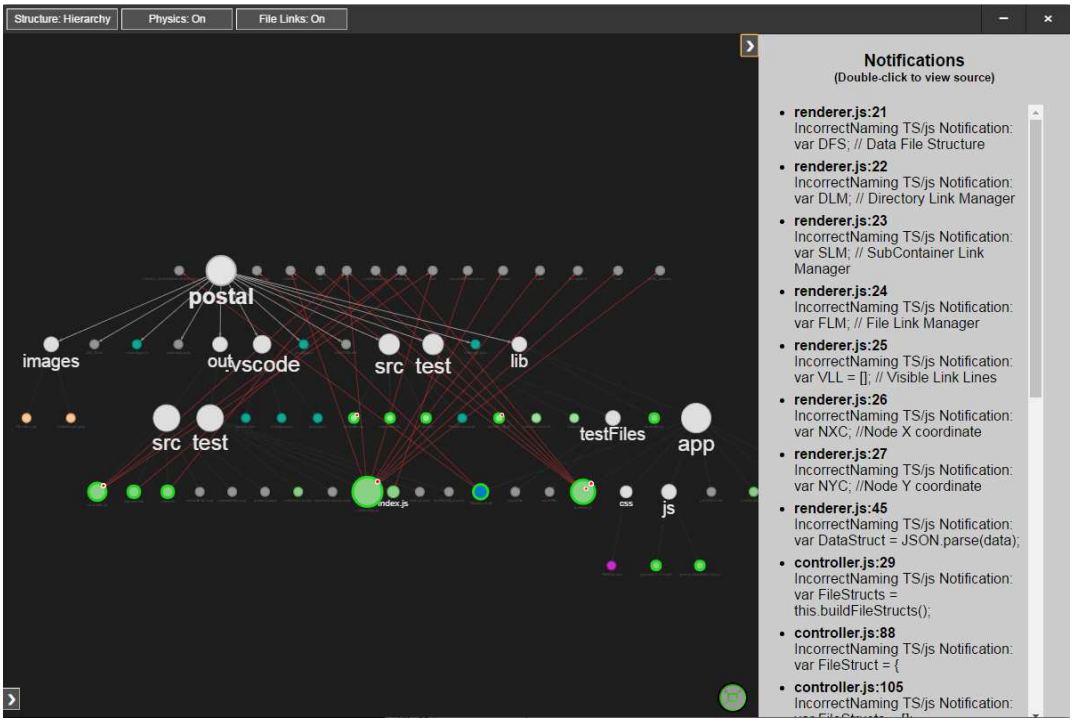

Figure 2. Notification Interface

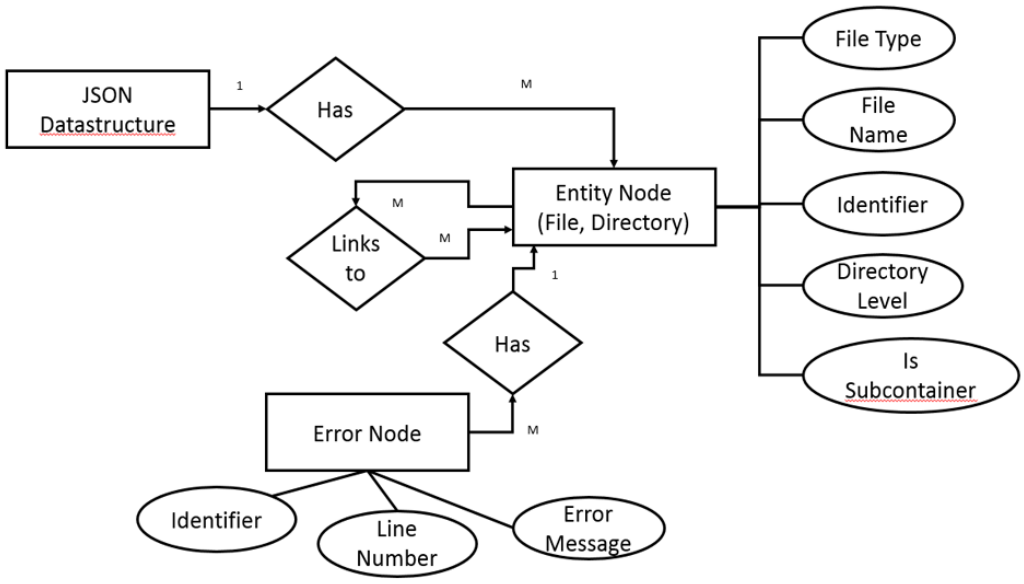

Figure 3. Updated Data Structure

IX. 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;
}
```