File References, Trees, and Computational Thinking

Craig S. Miller  Ljubomir Perkovic  Amber Settle  
DePaul University  DePaul University  DePaul University  
Chicago, IL 60604  Chicago, IL 60604  Chicago, IL 60604  
cmiller@cs.depaul.edu  lperkovic@cs.depaul.edu  asettle@cdm.depaul.edu

ABSTRACT

We study student understanding of the use of a tree structure in the context of an introductory web development course. In particular, we analyze student answers as they use a tree structure to construct file references in web pages. More fundamentally, our study initiates a bottom-up study of computational thinking by identifying the computational thinking mistakes that students make when they are learning resource referencing for web development. Our preliminary results suggest that students do not necessarily learn abstract concepts (like trees) and abstract rules of reasoning (composing relative and absolute tree paths) by just working with folders and composing file references alone.

Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computer and Information Science Education

General Terms

Design, Human Factors, Languages

Keywords

computational thinking, file referencing, tree structures, web development

1. INTRODUCTION

Computational thinking as a term has been defined recently by Wing [11]. It can be defined as the intellectual skills that a person uses to apply computational concepts and techniques to problems. Wing argues that: “Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child’s analytical ability”. [11]

Although computational thinking has been defined only recently, there are a number of projects with a focus on understanding computational thinking. One such project is “Computational Thinking across the Curriculum”, a project funded under the NSF CPATH program in which the authors of this paper are involved [10]. There are many other projects that are working on bringing computational thinking to the undergraduate curriculum. Some projects [1, 4] focus on creating courses for particular disciplines such as the sciences and engineering. Other projects [3, 2, 9, 5] work on integrating computational thinking across the entire undergraduate curriculum including our project. Our project is distinguished from other similar projects in that it focuses on integrating computational thinking into existing, discipline-specific courses.

If we are to effectively teach computational thinking skills to students, we will need to understand the intellectual processes at play when doing computational thinking. More pragmatically, just as we already do for reading, writing, and arithmetic, we will need to understand the computational thinking pitfalls and most effective ways to move students beyond them.

In this paper we consider student learning of tree structures for folder and file organization in a file system and student learning of absolute and relative references for locating files. Tree data structures and absolute and relative references for locating data stored in the tree are fundamental computational thinking concepts. We consider the learning of these concepts in the context of a DePaul University introductory web development class, IT 130: The Internet and the Web. IT 130 is one of the courses used by “Computational Thinking across the Curriculum” team members to teach computational thinking skills to non-computing majors. We are particularly interested in the mistakes students make when referencing resource files such as web pages or images. We categorize the mistakes made by students and interpret what they mean with respect to the teaching and learning of computational thinking rules of reasoning.

2. FILE REFERENCING

At some point any course on web development or design must cover how to reference a resource file such as a page link, style file or embedded image. For example, students need to learn how to code a working link by specifying the location of the page that appears when the link is selected. Typically an absolute reference is used for a page on another web site and a relative reference is used for a page that is part of the same web site. Learning the differences between absolute and relative references and their respective benefits typically coincides with instruction on coding links.

Despite being an elementary topic, the distinction be-
between absolute and relative references is important in many areas of computing such as memory addressing, spreadsheet references, and graphics. Early practice with the concepts in an introductory class may facilitate understanding in more advanced classes. Moreover, assuming a file structure is presented as a tree, students potentially learn how to traverse an abstract representation and apply it to a concrete application. Identifying misconceptions and correcting them at an early stage may provide positive transfer of knowledge to more advanced data structures learned later.

The topic and problem for this paper is presented as a tree structure. Figure 1 shows the file structure used for the exercise we discuss here. By presenting this tree as a file structure, we can ask students to provide a specific absolute reference or a relative reference given a current file.

For our current study, we focus on the file path and do not include the specification of the protocol (e.g., http) and the hostname from the reference. Without the protocol and the hostname, this reference is sometimes called the root-based reference. An example of an absolute file path to the *author.jpg* file would be `/project/images/author.jpg`. A full, absolute URL can be created by adding the protocol and hostname to the beginning of the absolute file path.

Relative references always require a starting file from which the relative reference originates. For example, a relative reference to the *author.jpg* given that the current file is *survey.html* would be `images/author.jpg`. Despite the focus on file references, our characterization of the task nevertheless requires students to apply core computing skills including: tracing through an abstract representation, employing symbolic notation that specifies the trace, and mapping an abstract representation to a concrete application.

### 3. STUDY OF STUDENT ANSWERS

Our initial study explores the mistakes students make as they work out the references to style files, image files and linked pages. The goal is to understand the range of errors and identify potential patterns for further investigation. To this end, we conducted a content analysis of student responses to an exercise. Rather than scoring results for quantitative comparisons, this exploratory stage calls for an in-depth analysis of student responses, which includes identifying common errors and potentially grouping them with the aim of proposing instructional strategies that address student misconceptions. Like other methodologies for content analysis [6], our approach involves a bottom-up analysis with the goal of producing clear classifications of responses that could be replicated in other studies. Our detailed analysis with a small number of students (9) produces some preliminary frequency counts, which are then supported with results from a much larger set of students (95).

#### 3.1 Method

Students in an introductory web programming class were asked to complete an exercise where they provide requested file references. This introductory course consists mostly of students from non-computing majors who take the course for liberal studies credit, although a few students are majors in computing programs such as computer science, information systems and information technology. Since the beginning of the term, they had already used both absolute references
and simple relative references. For the most part, absolute references could be correctly specified by copying the address from elsewhere. Relative references typically applied to files within the same folder as the current page. More recently, students were presented with examples where the relative reference was in a subfolder or required a path through a parent folder. While they were taught the notation using two dots (..) to reference the parent folder, all examples were given using diagrams of folders and not as tree structures.

For the study exercise, students were presented with the tree structure presented in Figure 1. It was introduced with a few sentences explaining the structure in the figure:

The diagram below represents a folder structure of a set of web files. The arrows indicate which items are contained within a particular folder. For example, the file quiz.html is contained inside the project folder.

The exercise then provided a few examples including those using an absolute file path and two relative references, one of which used the double-dot notation to access a parent folder.

The exercise then asked students to provide file references, of which three were absolute file paths (root-relative references) and five were relative references. Here are two examples from the exercise:

Q3: What is the absolute reference to the banner.jpg file?

Q4: What is the relative reference to the home.html file if the current page is bio.html?

All students turned in their answers to the exercises, but answers were only analyzed for the students who agreed to participate in the study.

### Table 1: Error Frequency

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute instead of relative</td>
<td>8</td>
</tr>
<tr>
<td>Indeterminate</td>
<td>1</td>
</tr>
<tr>
<td>Named current folder</td>
<td>7</td>
</tr>
<tr>
<td>Listed all names</td>
<td>9</td>
</tr>
<tr>
<td>Missing needed ascension</td>
<td>14</td>
</tr>
<tr>
<td>Used unneeded ascension</td>
<td>12</td>
</tr>
</tbody>
</table>

### Figure 2: List of Errors

<table>
<thead>
<tr>
<th>Error Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute instead of relative</td>
<td>../bio.html/home.html</td>
</tr>
<tr>
<td>Indeterminate</td>
<td>no answer</td>
</tr>
<tr>
<td>Listed all names</td>
<td>no relation to question</td>
</tr>
<tr>
<td>Missing needed ascension</td>
<td>no answer</td>
</tr>
<tr>
<td>Used unneeded ascension</td>
<td>no relation to question</td>
</tr>
</tbody>
</table>

### 3.2 Results

Nine students in the web course agreed to have their answers reviewed for the study. On average, 3.7 of 8 questions were completely correct. The answers were qualitatively analyzed by classifying and naming all errors. Figure 2 presents the exhaustive list of mistake names and the descriptions for identifying them in the student answers.

The same question could elicit a variety of errors from different students. For example, question 4 (relative reference to home.html if current page is bio.html) had a simple answer: home.html. However, only 1 of 9 students responded with the correct answer. Three students used the unneeded ascension operator (home.html), two provided absolute instead of relative paths (/home.html), one listed all names (bio/rootfolder/home.html), one used current instead of target (/bio.html) and one provided an answer, which seemed to have no relation to the question (images/globe.jpg).

After all errors were identified, error types were counted to identify common mistakes. Table 1 presents the frequencies for the conceptual errors, which exclude those labeled Notational dots, Missing answer, and No relation to question (assuming student answered wrong question). In a few cases, some answers qualified as multiple errors. For example, one student committed a Listed-all-names error 5 times, which also counted as 5 Named-current-page errors and 5 Named-current-folder errors.

The second column in the table shows the count of students making each mistake. Here a student was counted if that student made the corresponding mistake at least once. The third column shows the total frequency of mistakes across all students, noting that one student can make the same error multiple times.

The remaining columns reflect logical groupings of the er-
errors. For now, we focus on surface similarities between errors. In the next section we will discuss some reasons behind them and possible strategies for addressing them. A large number of errors involved level mismatch errors, which included cases where the reference went up one level too many and needed to go up one level but did not. As already noted for question 4, 3 of 8 students employed the two-dot notation to move up a level even though the current file was in the same folder as the target file. Less frequent but still repeated among students was the incorrect use of the opening slash. Again for question 4, two students started their path with an opening slash, which indicates an absolute file reference, even though the question called for a relative reference. Since all of these errors suggest difficulty in identifying the correct starting point for constructing the file path, the table groups these errors as initial anchor errors.

### 3.3 Supporting Results

Our study using nine participants permitted a detailed analysis of student errors. To verify whether these errors were representative of a larger population of students, we examined answers from an additional 95 students from six classes. Three classes were from the same course (IT 130) as our initial study and three classes were from a course emphasizing web design and human-computer interaction (HCI 201: Multimedia and the World-Wide Web). As part of a different study, the students were given a variety of preparations before they answered the path questions. The same questions were used, although the tree was expanded to allow additional examples and problems.

Errors were classified using the mid-level categories from the initial study. Correct answers (37.1% of all answers) and answers with multiple errors (32.6%) were counted separately. The remaining errors are listed in Table 2. The error type is noted in the first column. The second column shows its relative occurrence among all questions. The third column shows its relative occurrence among classified errors, which is useful for comparing percentages in Table 1.

Consistent with the initial content analysis, level errors, absolute/relative errors and use of implicit label errors were the most prominent, albeit with differences in relative order. As before, question 4 (requiring a simple relative path) yielded a disproportionate number of incorrect answers, with only 21% (20 students) correctly answering it.

### 4. DISCUSSION

Our initial study identified common errors and our second data collection provides supporting evidence on the frequency of their occurrence. We use the most frequent errors to direct our analysis and identify likely sources of misconceptions. Of particular interest is the observed difficulty in specifying the first part of a path, which we have grouped as initial anchor errors. To better understand what underlies these errors, we focus on question 4, which exhibited a variety of these errors.

At first glance, student difficulty in answering question 4 is surprising. Successfully naming the path simply requires specifying the file name in the folder. Anecdotally, instructors report little difficulty in getting students to link to files from the same folder. However, this question was among the most difficult in the context of the tree exercise. Further reflection suggests two possible reasons for this discrepancy. First, the exercise presented a number of examples and questions at various levels of complexity. It is possible that students were anticipating a more complex answer for question 4. Second, the file structure was presented as a tree while students only had experience of working with files in folders. It is possible that students could do the exercise well in the context of working folders but that the tree structure is too abstract.

Difficulty working with abstract representations is well documented in many studies, which has led many researchers to conclude, albeit controversially, that they are inherently hard to learn [7]. In this case, difficulty may arise from how a tree depicts sibling relationships. In particular, students may not readily conclude that two files with the same parent folder are actually contained in that (parent) folder. This failure may compel the student to unnecessarily ascend a level to access a sibling file. In contrast, viewing an actual folder may lead a student to see the correct relationship between two files in a folder. Consider Figure 3, which depicts the same project folder from the file structure in the tree representation. The folder representation clearly shows that these files and folder are contained within the same folder. While the tree structure presents the same information, a
student must trace through two tree vertices before drawing the same conclusion. This needed traversal may erroneously suggest to the student that additional notation is needed to construct the file path.

With respect to learning file references, our analysis suggests two strategies for addressing student errors. First, instruction should emphasize the initial construction of the file path, paying specific attention to the starting point within the tree structure. Almost all student errors involved the first element of the file structure and focusing on the relationship between the file component and the initial file reference might address these errors. Second, practice with the tree structure and its relationship to the folders might address the difficulties working with the tree representation. In particular, identifying sibling relationships in a folder and then within a tree structure may help secure a better understanding of the mapping between a concrete representation and the more abstract one.

Our experience suggests that the IT 130 students’ mistakes working with trees are not so much due to the difficulty of absorbing abstractions, but rather due to their general lack of familiarity and experience working with them. Most students in the class are not computer science majors and have not seen trees as an abstract data structure concept before. They are not taught about trees explicitly in IT 130 either. What our study suggests is that a substantial number of students did not learn about trees through the process of learning about the structure of folders in a file system and composing references to files. As mentioned already, this is not because they have not absorbed the concept of folders or how to compose references to files and URLs: in fact, they had already successfully created working references for their course projects. So, while they are able to navigate a file system, this knowledge does not translate to an understanding of a tree data structure as an abstract concept. We go further and venture a conjecture that these students are unlikely to apply tree abstractions and describe data location paths in other situations unless they are in well-familiar contexts such as, for example, a family tree.

As noted by Nisbett [7], many American psychologists and philosophers have for most of the 20th century doubted that an abstract rule of reasoning, such as the law of large numbers or a some rule of propositional logic, can be taught in its abstract form. They understood learning as an inductive process, a slog through a myriad of examples using the abstract rule. The collection of articles in the edited volume by Nisbett [8] painstakingly counters this thesis and pro-

vides strong evidence that abstract rules of reasoning can and should be taught directly, in their abstract form.

If we are to teach computational thinking – including the necessary toolkit of core computational concepts such as trees – to non-computing majors in context, the findings by Nisbett [8] (and supported by our preliminary study) show that it is critical to explicitly introduce abstractions and the abstract rules of reasoning (i.e. computational thinking) into the context. The context, while being the primary object of study of the class, then implicitly becomes a vehicle to illustrate the abstraction and the abstract rules and a vehicle to teach the student ways to encode situations into forms that are amenable to abstract rules. We argue that in the case of the IT 130 class, the computational concepts (tree data structures) and the computational thinking (say navigating trees) should be explicitly discussed. This may have the potential of not only helping the student within the context of web development but, more importantly, in future situations they will encounter.

5. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 0829671.

6. REFERENCES


