

Markup Language Diagramming: A Method to Enhance Web Page Design

William J. Tastle
Erica F. Hackley
Stephen W.J. Tastle

School of Business
Ithaca College
Ithaca, New York 14850-7170

Abstract

Web site design and creation, let alone the writing of the actual script that generated the web site, is difficult for the student who has minimal computer skills and has not benefited from a course in structured programming. Using the structure diagram method of Nassi and Shneiderman we develop a method by which web pages can be designed in such a way that the resulting hand-drawn image is used as a road map for the writing of the script. The method presented is called markup language diagrams and is represented by thirteen simplified symbols used to represent common web page structures.

Additionally, students have a difficult time finding and correcting their own HTML script when they do not have the benefit of application software such as DreamWeaver or Homesite where a formal indentation method is presented. Though simple to use, students should be exposed to a scripting standard as early as possible in their web site scripting experience to insure the development of proper scripting habits. A simplified scripting standard is proposed.

Keywords: Structure chart, Nassi-Shneiderman, HTML script, markup language diagram, MLD

1. INTRODUCTION

It has been observed over the past four semesters that there exist some general pedagogical problems associated with the instruction of HTML and JavaScript. Students possessing a background in coding, regardless of the language, are able to approach the generation of HTML script in a much more logical and constructive manner. Even students who reported taking a modest Web-weaving course in High School had a better grasp of the art of design and, consequently, more acceptable HTML script. About half of the students who possessed little or no previous background in Web design had much of their time and effort taken up by the design process of scripting, and then were challenged to produce acceptable and workable HTML script.

The problems appear to be centered in two distinct areas: *first*, while students generally have some idea of the kind of Web page they eventually want to produce, they have difficulty in describing it a manner such that the script can be properly written (Holzschlag 2001); and *second*, the writing of proper script done in a fashion such that it can be easily revised by the script author, as well as read by others, is a continuing source of a vast majority of scripting problems. Software tools exist to concurrently solve both problems, and they succeed and fail. Success comes from the eventual generation of a Web site that

looks good, but the underlying script that generates the image is all but unknown to the student (Coffee 2002). Thus, the student is unable to make even the simplest of changes to the underlying script. Also, some software packages produce script that has little or no structure, or at best an inconsistent structure, that belies its revision. One of the more popular and effective Web tools is Dream Weaver, and we have found it to be very effective, but only after students have learned to write script the old-fashioned way – by hand.

The purpose of this paper is to identify some common student problems and offer a novel solution to facilitate the design and scripting of Web sites. The data is derived from twelve sections of the introductory MIS course spread over three years. The course is split into three major components: the basic theoretical systems material characteristic of all MIS courses, a component in business process modeling, and a component in Web site analysis, design and creation. It is expected that no business student should graduate without having a basic understanding of the design and creation of Web sites sufficient to allow for the proper managing of others. We focus on this third component.

2. STUDENT SCRIPTING AND DESIGN PROBLEMS

We first address the easier of the two problems: the instruction of students to write script that can be easily revised. The motivation for this problem lies in the recollection of "spaghetti-like code" characteristic of first generation basic programmers. Recall that the first arguably popular programming language was BASIC, for which lines of code were distinguished by the placement of a unique integer beside each line of code. Control was passed from one section of the program to another by means of the notorious GOTO statement. This resulted in code that was all but impossible to understand, let alone manage. This precipitated the paradigm shift to structured programming languages and the introduction of procedures and functions to accompany a set of programming rules. Every programming language has adopted those rules of programming and students are taught, from the beginning of their programming career, to code following the rules. An example of a comparison of spaghetti code with structured code follows. A snippet of code to calculate the sum of a number of integers, written in some unstructured programming language, could be:

```
10 print "enter the number:"
20 input number
30 total = 0
40 counter = 1
50 total = total + counter
60 counter = counter + 1
70 if counter > number GOTO 90
80 GOTO 50
90 print "the sum is: "
91 print total
```

Notice that the underlying structure, that of a loop, is not immediately apparent. It takes a moment to study the code to determine its meaning. If this were a program of some 1,000+ lines of code, such a task would be formidable. The same problem written in a structured language could appear as:

```
"enter a number: " number
total = 0
counter = 0
while (counter = number)
{ total = total + counter
  counter = counter + 1
}
"the sum is: " total
```

It is conspicuously evident that a loop structure guides this code, and the structure is immediately visible. It is this same kind of obvious structured coding that needs to be used to instruct beginning students in the art of writing script. Over time, this kind of indentation has come into widespread practice in the programming discipline. In

today's introductory MIS class it is unfortunate that the majority of students have never been exposed to the concept of structured thinking.

To compensate for this increasingly important deficiency in business student thinking, we propose a standardization in HTML instruction in the intro MIS course, that of requiring students to use a plain editor, like Microsoft's Notepad, and displaying the resulting code in a browser. Both the editor and browser windows should be opened as side by side on the monitor. In this fashion, the student can quickly verify that the script written performs in the desired manner. However, and more to the point, the use of an editor forces the student to be attentive to the visualization of the script.

The following is an example of some script written in Notepad and using some simple HTML tags.

```
<body>
  <h1>
    Indenting The Code
  </h1>
  <hr width="50%">
  <ol>
    <li>
      It makes the code easier
      to read
    </li>
    <li>
      It helps to have it
      organized when one must
      troubleshoot problems.
    </li>
    <ul>
      <li>
        One can see if the
        proper tags have
        been closed
      </li>
      <li>
        It is easy to see if
        one instruction has
        been nested in
        another
      </li>
    </ul>
  </ol>
</body>
```

Notice that the indentation makes the intent of the script obvious and easily modifiable. This brings into focus the rules of writing HTML script:

1. Tags that do not have a closing tag, like `
` or `<hr>`, should be placed on its own line.
2. All tags that have closing tags, whether optional or required, must be required. Thus, tags such as `<p>` are required to end with the `</p>` tag, and the `` tag must also have the `` tag. These tags are

called encasing tags for they may contain one or more sets of other tags.

- Unless the content of a line is very short, the opening tag and closing tag must be aligned directly over each other. Whatever tags or content that is included within those tags is indented three spaces from the encasing tag.
- Tags that are identical to the encasing tags are called nested tags.

Without the required indentation of the script, the above example could appear as:

```
<body>
<h1>Indenting The Code</h1><hr
width="50%">
<ol><li>It makes the code
easier to read</li>
<li>It helps to have it
organized when one must
troubleshoot
problems.</li><ul><li>
One can see if the proper tags
have been closed</li><li>It is
easy to see if one instruction
has been nested in
another</li></ul></ol>
</body>
```

This kind of script is reminiscent of the spaghetti code of the past, but it should instead be called Monet script, a bunch of tags and text mixed up but delivering a visually appealing product. The indentation and alignment of script should be a requirement for successful completion of any student project.

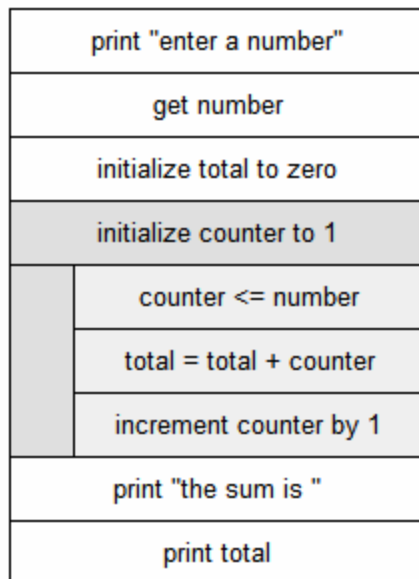


Figure 1 Previous example code as N-S diagram.

3. DESIGNING SCRIPT BY DESIGN

The ability to design a site and have that design also suggest the coding solution is another important element of HTML development. Among the more common problems we have noticed, one involves the generation of nested tables. Students appear to have difficulty because they are often confused in identifying which tag should be closed. For some individuals, it is difficult to visualize how the code being typed will translate to an actual web document. This causes them to prematurely close the first table before beginning the second because it is difficult to apply the concept of nesting, that is, the placing of a new table within a table detail <td> of a larger table. Students become frustrated and either produce poorly designed websites or they completely avoid the use of nested tables. Consequently, design fails and a lesser website is produced. If students had a tool by which they could visualize the format of nested tables, for example, we are convinced that more students would be able to grasp the nuances of the concepts and use them to produce websites would more dramatic and effective. What is required is a visualization tool that helps students to both create a website as well as assist in the formulation of the script. Fortunately, a method does exist which provides a basis for the creation of such a tool, and the method was created to assist in the skill of structured programming in computer science.

Figure 1 shows the code for the previous example as a Nassi-Shneiderman (1973) structure diagram. We note that this diagram is also referred to as an N-S diagram, structure chart, structure diagram, and structograph, the last being the term commonly used in European publications. It is easy to see the presence of the loop structure.

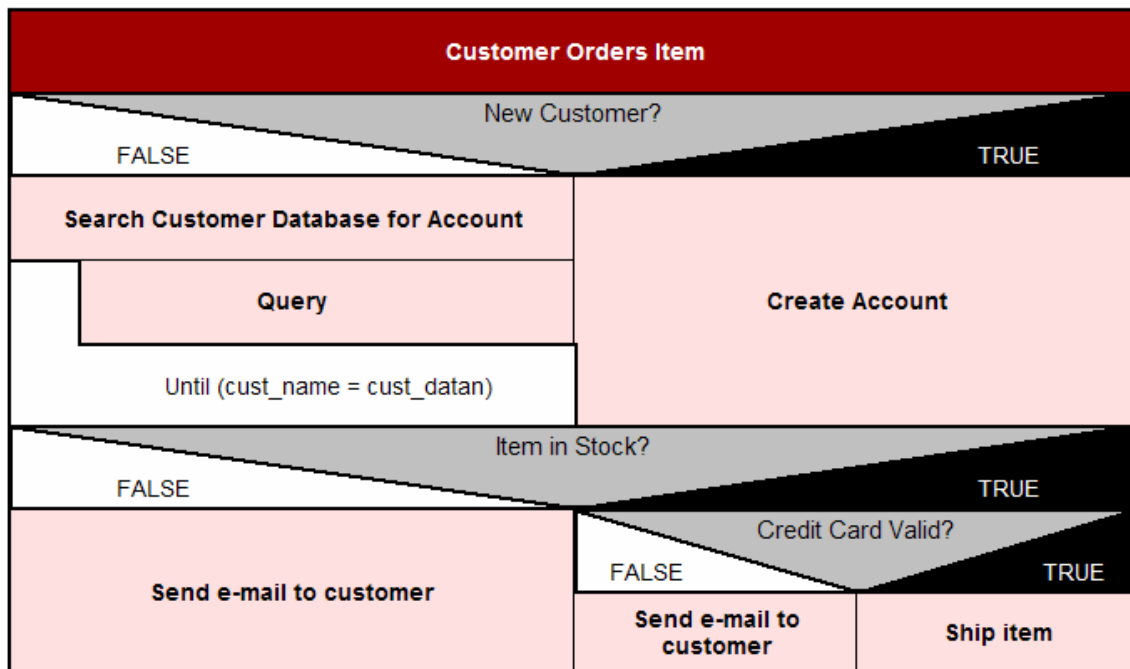


Figure 2 Example of a structure diagram showing decision structures and loop.

The structure chart is one of the first mechanisms developed for instructing students to design code in a structured manner was presented in a paper by Ike Nassi and Ben Shneiderman in 1973. Through a small set of visual structures students were able to design programs that would (almost) work the first time they were coded. Another example of an NS structure diagram is shown as figure 2. The general structures depict sequence, iteration, and decision programming structures. The sequence structure is certainly the most common for it represents a single line of code that is read and executed by the computer compiler once, and then control moves to the next line. The iteration structure is far more visual in that it allows students to "see" what portions of their code are repeated, the condition that eventually causes the loop to terminate, and how the content of variables is expected to change while the structure is iterating (looping). Finally, the decision structure allows for the clear and unambiguous visualization of the change in control given an evaluatory condition that is either true or false. Many software applications have been written to support this method, and a surprising number of them are in German.

In a manner similar to that of these structure charts, a method has been developed to provide a similar visual model of a website and the script that is used to create it. Without prior planning, students have problems with the website layout because it is hard for them to

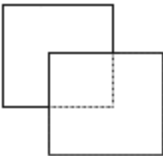
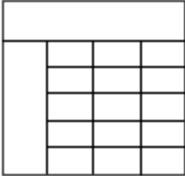




make the transition from the desired visual output to the written script. This can be related, at least in part, to the fact that we are a highly driven visual society in which it is much easier for people to work from pictures and create a verbal representation of the problem than it is to develop a visual image from a verbal description. An increasing number of MIS introductory courses (from: private communications with conference colleagues) are requiring students to write code to create web pages utilizing sounds, colors, images, animations, etc. Because it is easier for students to make the transition from pictures to words, we have developed a tool to help graphically depict the final outcome of the web page.

3. THE MARKUP LANGUAGE DIAGRAMMING LANGUAGE

Following the basic concept of Nassi and Shneiderman we have sought to develop a method by which a website can be drawn out as many students typically do before writing script. We have added to the diagram by including symbols that represent certain basic, commonly used hypertext tags. We intend for it to simplify and shorten the learning process of creating web pages by graphically organizing the outline of the web page from which we can easily convert the diagram into code. We have created Markup Language Diagrams (MLDs) to be

logical, structured, visually understandable, and universally recognizable. The symbols that have been developed for the MLDs are easy to remember and simple to draw, thus allowing website designers to easily outline the information they want to present.

We have currently developed 13 symbols which represent the most used aspects of a website. These are:

Construct	Graphical Representation	Explanation
Rollover		Two squares, with one slightly covering the other symbolizing the effect of a <i>mouseover</i> . The names of the pictures are recorded within the squares.
Table		Intersecting vertical and horizontal lines represent a table. Each "box," an intersection of column and row, represents an individual cell within the table. <i>Cols</i> and <i>rows</i> can be represented in the table. If the outline of a table is in doubt, it may be heavily bolded to make it stand out.
Textbox		A rectangle with an asterisk in the upper-left hand corner. The asterisk indicated that the user adds text.
Radio Button		Small circles aligned in either a vertical or horizontal format.
Pictures/Graphics		A rectangle with corners blackened. This symbol represents the placing of a photograph into a photo album that is being held down by corner tabs. The name of the image is recorded within the rectangle. In the event the name and path is too long to appear in the symbol, an integer is placed in the symbol and the corresponding image name and path included as an addendum.
Paragraph of Text		Represents the presence of a text paragraph and is followed by a number to use as a reference since the text can be on a separate page.


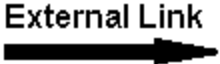





Internal Link		Similar to that of the external link (see below), but has an extra arrow pointing either up or down to point to the link location.
External Link		The image name (text, picture, etc) with an arrow underscoring it pointing to the right represents the linking to another page.
Frames		Two rectangles, one nested within the other, symbolizing a picture frame.
Forms		A square with a diagonal line in the upper left and lower right hand corners to represent a form (the beginning and end symbolized by the diagonal lines). The internal content is represented by other symbols. This symbol can become quite large to accommodate the other symbols.
Dropdown Menu		A horizontal rectangle placed above a vertical rectangle represents a drop down selection box. In the horizontal box is placed the name of either the default item or the title of the options listed in the vertical rectangle. Into the vertical rectangle can be placed either the options or a number showing the details in the addendum.
Command Buttons		A triangle represents a command button. Inside is written the command to be performed (ie submit, reset).
Checkboxes		A square with a check located inside it positioned in either a vertical or horizontal line.

Figure 3 MLD symbols.

In addition to these symbols it is necessary to note some of the design instructions as parameters used in other tags. For example, one of the parameters of a table tag is the table width, expressed as a percent of the window or as a fixed number of pixels. MLD allows the placement of parameters directly on the design where it is first needed. In our example that appears as figure 4 we show the width of the web page at the every top of the page as a fixed 600 pixels. Since the Nassi-Shneiderman diagrams helped to structure the programming code and because designing code and web page design are similar, it is apparent that web page design could also use a little more structure. Hassell (1982) identified two major disadvantages to the Nassi-Shneiderman diagrams: one was their inability to be easily converted into a machine sensible form and the second was the inability to easily change any element of the diagram without the arduous task of redrawing it. The second disadvantage was eliminated with the advent of the applications that provide for computer-generated NS diagrams that are easily adjusted. MLD's are designed so as to avoid those two disadvantages, especially the second one since there is no computer tool to generate the diagram at present. Markup Language Diagramming is designed to include the major elements needed for the creation of a web page

4. AN EXAMPLE OF AN MLD

Using the symbols of figure 3, an MLD was designed to describe the all the tags and give a description of their usage and is available from the authors. The code required to generate this web page is also available. Examination of the code will show a strict adherence to the indentation standard and thus, it is easy to identify each aspect of the web page even without the presence of comments. The need to comment the script is greatly reduced, although definitely not eliminated, though for purposes of illustration of the indentation format no comments appear.

Students have been required to use this formatting standard in all their scripting assignments and have generally commented on the usefulness of indentation. Those who have had problems with the indentation are characteristically students who have learned to write script on their own and use virtually no structure whatsoever, and those who have taken an HTML class in high school. Our secondary school colleagues have not yet generally established the concept of structured code in instruction.

5. CONCLUSION

The transition from web site design to the generation of web script can be greatly improved through the introduction of a new design method called the markup language diagram. This formal model allows the student to draw out the conceptual web pages much as would be done without benefit of this method, but it includes the use of simple symbols that allows for the generation of a "road map" to the writing of the script. Further, while the

symbols are being placed into the web site design the student is able to logically plan out the operation and navigation of the page(s). The scripting details are remanded to another sheet where they are enumerated and described or listed. Further, the usage of indentation as a forced discipline allows students to write script that is more correct and gives them a way of identifying and correcting their own coding errors. Students need to be taught to write structured script from the first course in order for a habit to develop, much like the writing of structured code in any programming language. Those students who have embraced structure in their script writing produce better quality output, but that is a matter for future discussion.

6. REFERENCES

- Coffee, Peter, 2002, "Mad as Hell about 'Food Fight' HTML." eWeek, April 1.
- Hassell, Johnette and Victor Law, 1982, "Tutorial on Structure Charts as an Algorithm Design Tool." ACM 0-89791-067-2/82/002/0211.
- Holzschlag, Molly, 2001, "XHTML in the Real World." <http://www.webreview.com/>.
- Nassi, I. And B. Shneiderman, 1973, "Flowchart Techniques for Structured Programming." SIGPLAN Notices, vol 12.