

California State University, San Bernardino

**CSUSB ScholarWorks**

---

Theses Digitization Project

John M. Pfau Library

---

2004

## Web Texturizer: Exploring intra web document dependencies

Seema Amit Tandon

Follow this and additional works at: <https://scholarworks.lib.csusb.edu/etd-project>



Part of the [Computer Sciences Commons](#)

---

### Recommended Citation

Tandon, Seema Amit, "Web Texturizer: Exploring intra web document dependencies" (2004). *Theses Digitization Project*. 2539.

<https://scholarworks.lib.csusb.edu/etd-project/2539>

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact [scholarworks@csusb.edu](mailto:scholarworks@csusb.edu).

WEB TEXTURIZER:  
EXPLORING INTRA WEB DOCUMENT DEPENDENCIES

---

A Project  
Presented to the  
Faculty of  
California State University,  
San Bernardino

---

In Partial Fulfillment  
of the Requirements for the Degree  
Master of Science  
in  
Computer Science

---

by  
Seema Amit Tandon  
March 2004

WEB TEXTURIZER:  
EXPLORING INTRA WEB DOCUMENT DEPENDENCIES

---

A Project  
Presented to the  
Faculty of  
California State University,  
San Bernardino

---

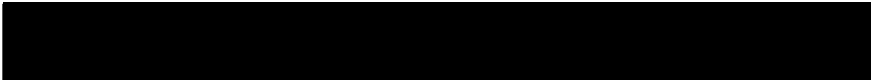
by  
Seema Amit Tandon  
March 2004

Approved by:

  
Kerstin Voigt,  
Department Of Computer Science

03/01/04  
Date

  
George Georgiou, Computer Science

  
Richard Botting, Computer Science

## ABSTRACT

The purpose of this Master's Project was to develop a customized browser to facilitate document skimming. Web Texturizer facilitates document skimming by visually emphasizing important text segments for the user through the user-friendly GUI and an Internet browser (Internet Explorer).

The Web Texturizer developed is interactive: as the user requests the web document, enters additional keywords and the keyword count, and hits enter, a CGI-generated analyzed web page will be returned. The user will be able to skim through the entire document by visually emphasized paragraphs and a tool to navigate through most related paragraphs. The algorithm uses a "soft" logic that incorporates a notion of textual similarity developed in the information retrieval community. Here, we explore a general-purpose method to automatically recognizing structure in HTML documents. Knowledge of both text structure and the informational needs and preferences of the user are employed to offset the viewed document to the most relevant sections of the text.

## ACKNOWLEDGMENTS

The National Science Foundation Research Career Integration Program (MI-I) Stipend provided a grant that I used for this project. The grant money was also used to purchase O’Rielly Perl Cookbook, JavaScript Cookbook and host the site with a web-site hosting company and print other research papers that are required for this project.

Thanks need to be given to the professors, who helped with this project. Dr. Kerstin Voigt, developed SKIPPER project which was the genesis of this project. Dr. Richard Botting’s help with the vast research papers for analysis and suggestions. Dr. George Georgiou and Dr. Richard Botting were two more instructors, who acted as advisors and helped a lot during the whole master’s process.

Of course, without the help and support of my family, I would have never completed this project. To my husband, Amit, my cute daughter Deepika and my brother and sister-in-law, Vinod and Jyoti, I love you.

## TABLE OF CONTENTS

ABSTRACT . . . . .	iii
ACKNOWLEDGMENTS . . . . .	iv
LIST OF TABLES . . . . .	vii
LIST OF FIGURES . . . . .	viii
CHAPTER ONE: SOFTWARE REQUIREMENTS SPECIFICATION	
1.1 Introduction . . . . .	1
1.2 Overall Description . . . . .	8
1.3 Specific Requirements . . . . .	17
CHAPTER TWO: NUMERICAL ANALYSIS OF WEB TEXTURIZER EQUATIONS	
2.1 Web Texturizer Equations . . . . .	25
CHAPTER THREE: DESIGN	
3.1 Architecture (Component Diagram) . . . . .	30
3.2 Detailed Design (Pseudo-Code) . . . . .	34
CHAPTER FOUR: OPERATING INSTRUCTIONS	
4.1 Operating Instructions . . . . .	38
4.2 Hints for the Users . . . . .	40
4.3 Testing . . . . .	42
4.4 How to Install Web Texturizer . . . . .	42
CHAPTER FIVE: MAINTENANCE	
5.1 Files . . . . .	45
5.2 Directories . . . . .	46

## CHAPTER SIX: WEB TEXTURIZER ANALYSIS

6.1 Web Texturizer Versus Subjective Opinion . . . . .	48
---	----

## CHAPTER SEVEN: FUTURE DEVELOPMENTS AND CONCLUSIONS

7.1 Ideas for Future Developments . . . . .	56
7.2 Conclusions . . . . .	57

APPENDIX A: WEB TEXTURIZER CODE . . . . .	59
---	----

REFERENCES . . . . .	89
----------------------	----

## LIST OF TABLES

Table 1.	Minimum System Requirements . . . . .	10
Table 2.	UrlText.cgi Class Diagram . . . . .	31
Table 3.	PageParser.pm Class Diagram . . . . .	32
Table 4.	Vector.pm Class Diagram . . . . .	33
Table 5.	Menu Class Diagram . . . . .	33
Table 6.	UrlText.cgi Pseudo-Code . . . . .	35
Table 7.	PageParser.pm Pseudo-Code . . . . .	35
Table 8.	Vector.pm Pseudo-Code . . . . .	36
Table 9.	Menu Pseudo-Code . . . . .	37
Table 10.	Perl Modules that Need to be Installed . . .	45
Table 11.	HTML Files . . . . .	46



## LIST OF FIGURES

Figure 1.	Diagram of CGI using Perl . . . . .	7
Figure 2.	Deployment Diagram . . . . .	12
Figure 3.	Use Case Diagram . . . . .	14
Figure 4.	Web Texturizer's Home Page . . . . .	17
Figure 5.	Web Texturizer Analysis Page . . . . .	18
Figure 6.	URL Analyzer Page . . . . .	18
Figure 7.	Web Text Similarity Analyzer . . . . .	20
Figure 8.	Web Text Analyzer . . . . .	21
Figure 9.	Component Diagram Overview . . . . .	30
Figure 10.	Table Containing Text . . . . .	41
Figure 11.	Web Texturizer Versus Subjective Opinion . . . . .	48

## CHAPTER ONE

### SOFTWARE REQUIREMENTS SPECIFICATION

#### 1.1 Introduction

##### 1.1.1 Purpose

The goal of this Master's Project is to create a customized web-browser to facilitate the skimming of documents by offsetting the document with relevant information. This project is built on the SKIPPER project and promotes active learning by providing users with a tool to navigate through the entire document based on automatically generated keywords. Using the WHIRL and Porter's Stemming algorithm generates these keywords (2). The objective was to develop a web-based application, which is easily distributed, flexible, intuitive and easy to use.

Moreover, the application was to adapt to users with varying degrees of familiarity or expertise in the subject area of the document being viewed. At the extreme ends of the spectrum, we distinguish novices and experts.

Traditional modes of viewing a document (scanning and scrolling) are supported for novices who need to see the information in its intended context. Experts with prior knowledge of the document or document subject will be able to skim and skip the entire document using the relevance

measures that are provided to determine frequencies of keywords among various paragraphs.

### 1.1.2 Scope

The Web Texturizer project is focused on developing a web based tool that uses Perl and Java-Script technologies to facilitate document skimming by visually emphasizing important text segments for the user through an user-friendly GUI and an Internet browser. The Web Texturizer project is focused on developing an intelligent agent analysis tool that helps guide in the presentation of information in ways that make personally relevant information more easily accessible to the user in the area of expertise of a particular document (9). The idea for the creation of this project came from Dr. Kerstin Voigt, Professor of Computer Science and her paper (9). The materials analyzed can range from research papers, tutorials, to literature of investment documents in order to skim through the web page. The materials developed are designed to skillfully guide the user through the site of personal interest based on their background and expertise.

The Internet address or URL for the current version of the Web Texturizer Project is:

<http://textseem.ehost4u.com/webtext-dev/index.html>. The

Web Texturizer, has an interactive navigational tool. Users can personalize a browser of large informational web page by (1) automatically analyzing and detecting keywords of apparent personal interest to the current user, and (2) by highlighting and presenting relevant information to the user in a structured textual fashion. Thereby the document is organized according to the relevance ranking of the keywords in the respective web page. The tool is designed such that the users can modify the keywords or the count based on their expertise. With the help of Dr. Kerstin Voigt, a Professor of Computer Science at CSUSB, this Master's Project added techniques of learning information retrieval to automate the web browsing experience to the Web Texturizer.

CGI generated Perl based programs can provide tremendous text processing capabilities and allow users with different interest to traverse through these information nets more frequently than others. The Web Texturizer is written in Perl, it has been written in an object-oriented manner, perl functions are modular and readily adapted to new uses. As a navigating tool, a well-written application will allow the users to focus attention on the material while minimizing the need to know the Perl

programming language. The objective was to produce an application, by developing general purpose methods for automatically recognizing structure in HTML document. The goal is to extract structures of information from web pages without any page-specific program training. In the Web Texturizer tool, WHIRL is used to semi-automatically generate keywords from structured documents. WHIRL stands for a Word-based Information Representation Language. This tool is implemented using WHIRL, a method that demonstrates a notion of textual similarity developed in the information retrieval community (3).

The main benefit is that customized browsing can provide speedier access of information based on user's expertise in the browsed subject matter. In a raw web page, it is likely that the user will have to spend more time in reading and determining if the web-page is relevant to the user's needs. Web Texturizer facilitates document skimming and aims to provide a one-stop source of document browsing instead of traversing through its myriad of text and hyperlinks. A major advantage of Web Texturizer is to skillfully highlight keywords and automatically and autonomously personalize one's interaction with the web page.

Also, another advantage is "individualization", that is, to help users cope with the high-volume of data made available by large informational web pages. The Web Texturizer is intended for users who repeatedly, regularly, and for an extended period of time, revisit and navigate the same information subspace. Finally, Web Texturizer assists the user in searching and navigating textual information. Hence, the ability to analyze text is the center of this project. Word frequencies and co-occurrences between text segments are mined. Text-formatting cues are used to identify distinct paths of the text and establish relationships between text segments.

### 1.1.3 Definitions, Acronyms, And Abbreviations

1.1.3.1 HTML. HTML is a document - layout and hyperlink -specification language. HTML stands for HyperText Markup Language. The language also tells how to make a document interactive through special hyperlink, which connect user document with other documents or other Internet resources. Perl and CGI are used in the Web Texturizer to generate dynamic HTML documents. HTML provides structured meaning to the content of web pages. For example, a piece of text can be made into a paragraph by the tags (<P>...</P>), or it may be tagged as bold

(<B>...</B>) to signal its significance. The scripts are written in Perl 5.6 are loaded on a server on the World Wide Web via a web browser and run inside an HTML web page. A web browser such as Microsoft's Internet Explorer can be used to run these scripts.

#### 1.1.3.2 Common Gateway Interface using Perl V 5.6.

CGI is an acronym for Common Gateway Interface, which defines the standard in which external programs should communicate with a web server. A majority of the CGI programs are written in PERL (Practical Extraction and Report Language) and the end result is an executable file that can read and write information in a format defined by the CGI protocol. Figure 1, describes the Perl CGI interface.

[1] With CGI, a HTTP request is sent via a web browser.

[2] The Web server can call up a program written in Perl, while passing user-specific data to the program (such as what host the user is connecting from, or the input the user has supplied using HTML for syntax).

[3] The program then processes the data and the server passes the program's response back to the web browser. In this project, CGI scripts use Perl's extensive

regular expression matching facilities through perl modules to provide fast key-phrase searches within the HTML document. The implementation of Web Texturizer benefits from extensive library of Perl Modules available at <http://www.perl.com>.

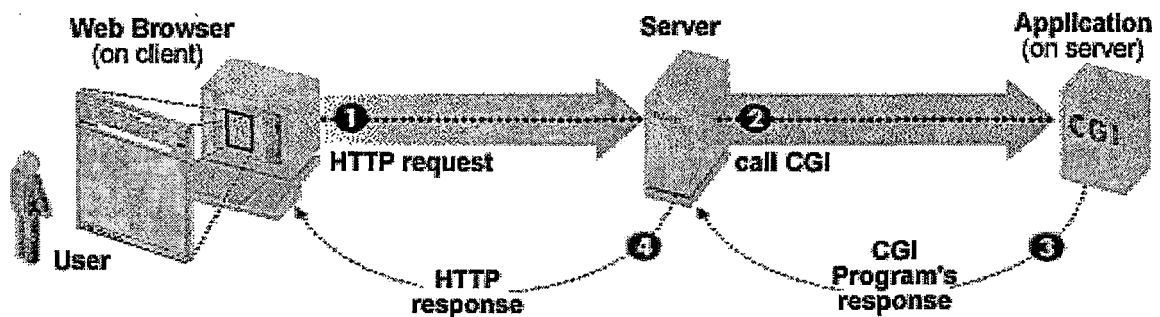


Figure 1. Diagram of CGI using Perl

There are several reasons for using Perl for processing text in the Web Texturizer tool. First, perl has a regular expression engine. This is a pattern matching language that is based on sed and awk, two text processing tools that are part of UNIX. Perl regular expressions are very efficient and very powerful. The second text processing mechanism in perl is the text processing functions that are built into Perl. Many of these functions are unique and do not exist in other popular languages. Some of these functions include



`split()`, `shift()`, `pop()`, `chomp()`, `join()` and `slice()`.

These functions along with Perl's use of dynamic strings, hashes (denoted as %), and arrays are extremely useful for text processing. The last text processing mechanism in Perl is external modules that can be loaded dynamically. Many of these modules themselves are built using regular expressions and Perl functions. Examples include `HTML::Parser` and `Lingua::Stem`. Modules are usually easier to use than regular expressions and are typically built to solve a particular text processing problem, such as parsing HTML files.

1.1.3.3 JAVASCRIPT. JavaScript is used for the implementation of components of the Web Texturizer. These scripts are used to automate certain HTML tasks. These can be coded within the HTML document and can be run on any JavaScript enabled web-browser.

## 1.2 Overall Description

### 1.2.1 Project Perspective

1.2.1.1 System Interfaces. Web Texturizer is installed on a server computer at <http://textseem.ehost4u.com/webtext-dev/index.html>. Users

are able to access the Web pages and run the CGI application with Internet Explorer.

1.2.1.2 User Interfaces. The Web Texturizer scripts can be run by entering the URL to analyze in the top frame. The analysis of the web page occurs in the bottom frame that displays the content generated by the URL entered on the top frame. The left frame consists of JavaScript code that enables the user to skip and scan the web page based on the paragraphs that have high similarities. The User Interface of this program consists of one web page that the user enters the URL to analyze. The execution of the program is completely keyboard and mouse pointing device based. All clicks and selections from the left frame and the content of the anchored links are accomplished with the click of the mouse button.

1.2.1.3 Hardware Interfaces. This project was written and tested on a SGI Computer, which supports a Unix operating system. This program may run on any Unix machine that has Perl 5.0 installed. The system specifications are:

Table 1. Minimum System Requirements

Processor Type	Intel MMX, Pentium II
Processor Speed	450 MHz
System memory	64 MB
External Cache Memory	256 KB
Modem	LT Win Modem

1.2.1.4 Software Interface. This project was designed to run over the Internet. By writing it in Perl, the code will run on any computer independent of the operating system. As long as the user's computer has an Internet connection, and appropriate browser, there will be no other limits on the users system.

Perl is one of the best programming languages for text processing using perl regular expressions (5). Perl allows you to rapidly design, program, debug, and deploy applications. Through the module importation mechanism, you can use these external definitions as if they were built-in features of Perl. Object-oriented libraries maintain their object-oriented structure in Perl. The Internet is growing exponentially and unfortunately has become one of the major battlefields in the computer world. Over the Internet, there are various systems providing services. Each platform has its own system development environment. Even in the Unix environment, different

vendors have their specially designed Unix systems. They are not compatible with each other. A common language was not a major issue in the past because the operating environment was not complex. However, with the growth of the Internet, a common CGI becomes more important. For this reason, Perl was developed.

1.2.1.5 Communications Interfaces. Figure 2, the Deployment Diagram, shows the relationship between the Web Texturizer, the network server and the Internet user. To access this site, the user needs an Internet connection to the Web Texturizer server. The web address is:

<http://textseem.ehost4u.com/webtext-dev/index.html>.

The user will also need a web browser, such as, Internet Explorer versions 5.5 or higher.

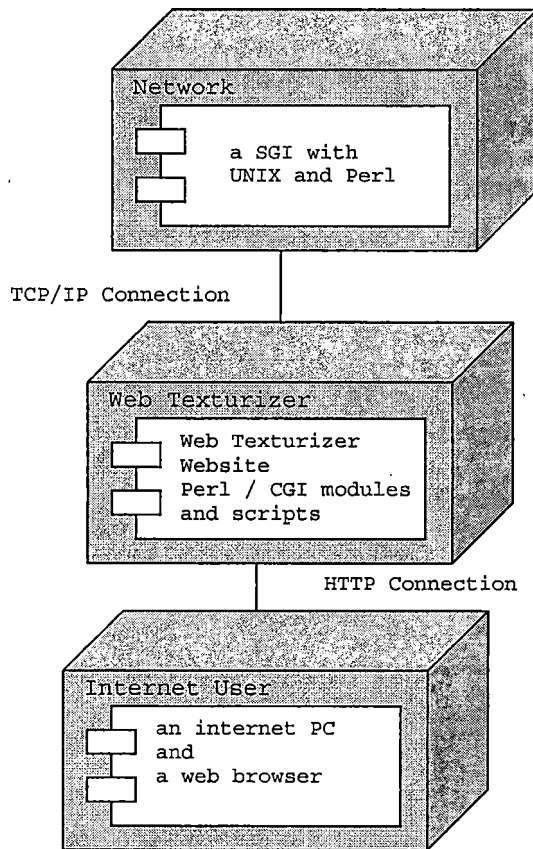


Figure 2. Deployment Diagram

1.2.1.6 Memory Constraints. The minimum RAM, Random Access Memory, tested, for this program, was 64 Megabytes. However, any computer with enough memory to effectively use the Internet should be able to use this program.

1.2.1.7 Operations. The Web Texturizer will be accessed on a server on the World Wide Web. It will remain active as long as the browser containing the URL is running

and the user has an internet connection. The program is interactive and needs the user to interface with it. Nothing happens unless the user enters a URL to analyze. Currently, the program does not use a database. Therefore, there are no backup or recovery operations required. Once running, the page should be re-created with keywords and users can skip and scan through paragraphs. If the page takes a long time to load it may be due to firewalls set up on the local server where the data is being accessed from or that it is not an HTML page but rather a PDF or post-script file. However, if a problem should occur, simply using the browser refresh button will reset the web page.

1.2.1.8 Site Adaptation Requirements. This project is being written and tested with an SGI machine with UNIX Operating System and Internet Explorer 5.5. Earlier versions of Internet Explorer have not been tested.

#### 1.2.2 Project Functions

The Web Texturizer has one main function. It is used to analyze the URL entered, record and guide the individual browsing activities of the user. Hence, this tool will adapt to users varying degrees of familiarity of a document while browsing across different subject areas. The tool allows a user who browses subject areas in which they have

little expertise, may need information presented as coherently as possible based on the keywords automatically generated by the tool. On the other hand, a user who considers himself an expert is quite capable of comprehending the presented information and relies on the percentage of occurrences of keywords among paragraphs. Figure 3, Use Case Diagram, shows the relationship of the user to the Web Texturizer.

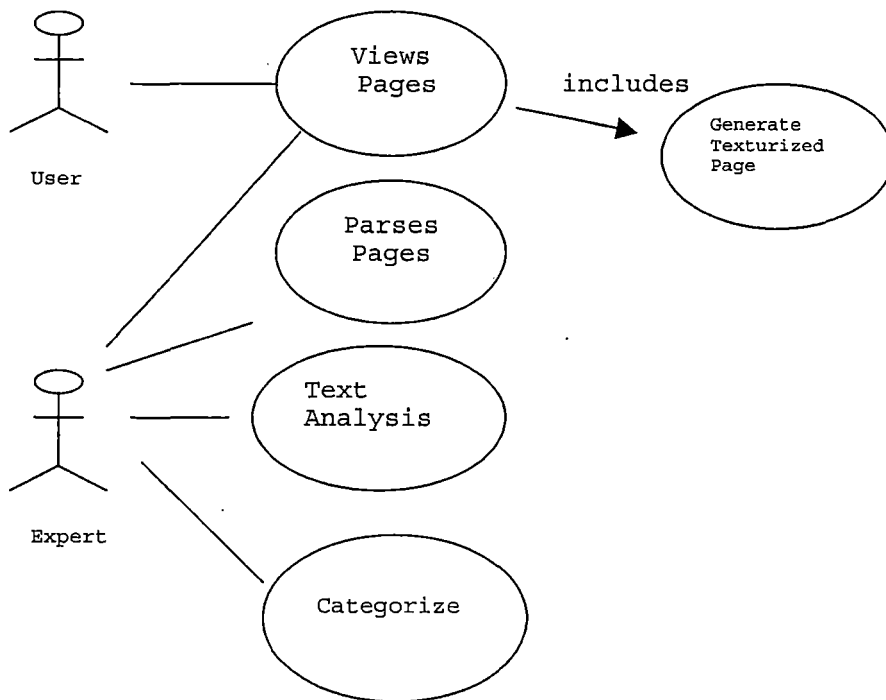


Figure 3. Use Case Diagram

This model displays how the components of Web Texturizer are organized. Hence, the user maybe an expert in the subject area browsed, and enjoys accelerated access of relevant information provided by a modified customized document. Also, the user can skip through paragraphs that are closely related. Alternatively, for a novice the browser may provide information comprehended best within the context of the original web document. Users can enter their own keywords along with the count of the keywords they want the tool to automatically analyze. They can use the Next and Previous buttons to navigate through various paragraphs that are closely related.

### 1.2.3 User Characteristics

The audience for this project is experts such as professors, and novices such as students. The students that use this site may be able to benefit from the fast access to the best paragraphs that they need to review. The professors may access this site because they want to review the most closely related paragraphs. As well as, review the percentages of paragraphs that are highly related. Any science or business majors can access this site and benefit from its document scanning capabilities.



The instructors who will use this project will fall into two camps: Computer Science and artificial intelligence agents literate and professors that want to read research papers in order to quickly scan the document. The tool is written in such a manner that no knowledge of Perl is required. If the user can open and run basic Internet browsers, they will be able to run the tool. The instructors and users literate in Perl will be able to modify and extend the tool to meet their own needs and build on them to write and create a database driven tool.

#### 1.2.4 Constraints

The Web Texturizer tools was developed and tested on Internet Explorer. A perl programmer can update the script to be user with other web browsers. Anybody, who would like to modify or create new Web Texturizer applications, will need to know how to program in Perl.

#### 1.2.5 Assumptions and Dependencies

The web server should not have restrictions on accessing other web pages. The web server should not have firewalls or password protection set on accessing other web pages. Firewalls and password authentication are frequently used to prevent unauthorized users from accessing private networks.

### 1.2.6 Apportioning of Requirements

In the context of this project, there are no elements that are being delayed until future versions are developed. The program code has been finalized at this time. However, there are several items that could be improved by another computer science student in the future.

## 1.3 Specific Requirements

### 1.3.1 External Interfaces

The tool can be found by going to the Web Texturizer server and loading Web Texturizer home page. The web page will provide instruction on how to use the tool. The text contents of the web pages are included in the appendix.

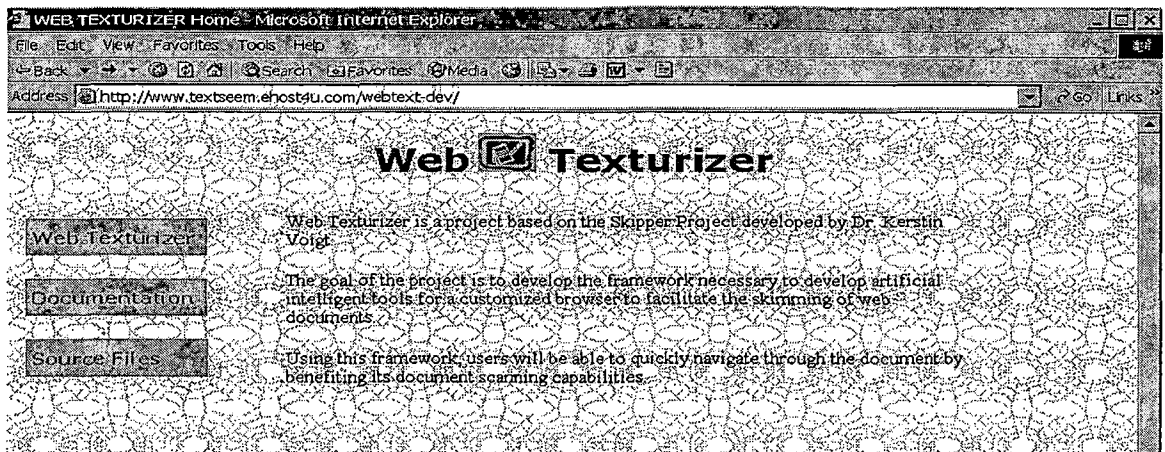


Figure 4. Web Texturizer's Home Page

The Web Texturizer Analysis page is displayed in Figure 5.

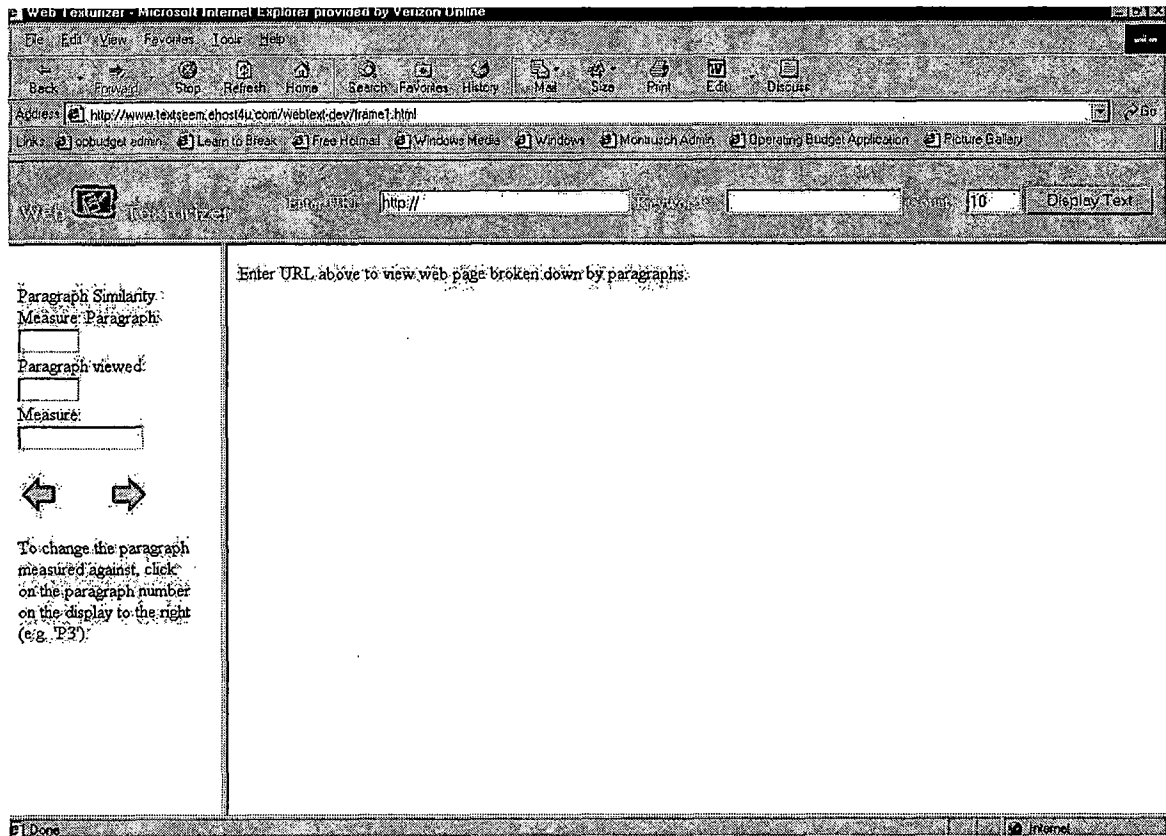


Figure 5. Web Texturizer Analysis Page

On the Web Texturizer page, the user enters the website that they want to analyze.

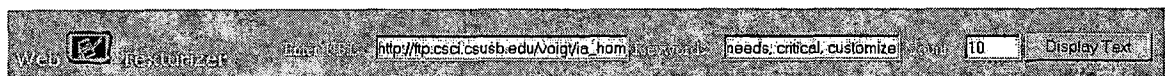


Figure 6. URL Analyzer Page

This application will use a frame, Figure 5, and is divided into URL bar and two panels. The main part of the

frame is divided into two unequal panels, split vertically.

The URL Analyzer bar contains 3 text boxes:

Enter URL: User inputs the URL that needs to be analyzed.

Keywords: 3 keywords separated by commas that user expects to be contained in the main contain.

Count: The number of keywords that user wants the Web Texturizer to count.

The left panel is the "Web Text Similarity Analyzer"

(Figure 7), consists of three text boxes: the first box displays the current paragraph, the second box displays the next most similar paragraph that is related to the first one, and the third text box displays the similarity measure with respect to the similar paragraph. This panel also includes a set of buttons, which enable the user to skip to 'Next' or 'Previous' paragraphs. The right panel is the "Web Text Analyzer" panel (Figure 8). It displays the analyzed text document along with the highlighted keywords and links to other related paragraphs.

Paragraph Similarity  
Measure: Paragraph  
  
Paragraph viewed:  
  
Measure:

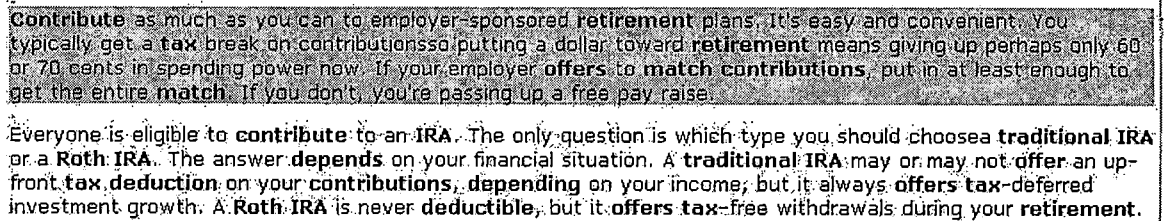
← →

To change the paragraph  
measured against, click  
on the paragraph number  
on the display to the right  
(e.g. 'P3').

Figure 7. Web Text Similarity Analyzer

The first step is for the user to enter the URL in the URL Analyzer Panel. Once the user enters the URL, the user can enter the 3 keywords and count of the automated keywords. After all the data is entered the user can either click the mouse button or use the enter key to view the results. Once the user enters the URL, complete text document along with the highlighted keywords is displayed in the right panel. The user can navigate through the different paragraphs using the next and previous buttons on the left panel. The user can also view the similarity

measures of the different paragraphs.



Contribute as much as you can to employer-sponsored retirement plans. It's easy and convenient. You typically get a tax break on contributions so putting a dollar toward retirement means giving up perhaps only 60 or 70 cents in spending power now. If your employer offers to match contributions, put in at least enough to get the entire match. If you don't, you're passing up a free pay raise.

Everyone is eligible to contribute to an IRA. The only question is which type you should choose a traditional IRA or a Roth IRA. The answer depends on your financial situation. A traditional IRA may or may not offer an up-front tax deduction on your contributions, depending on your income, but it always offers tax-deferred investment growth. A Roth IRA is never deductible, but it offers tax-free withdrawals during your retirement.

Figure 8. Web Text Analyzer

The Web Text Analyzer (Figure 8) displays the web page and highlighted keywords. If there is a hyperlink within the web page that is clicked by the user, the functionalities of the Web Texturizer tool are propagated and applied to the linked document as well.

### 1.3.2 Functions

One of the functions that needed to be defined is how the program accepts and processes its inputs and outputs. The only input, once the website is running on the user's browser, is the mouse and the keyboard. Therefore, there is no requirement to check the validity of the inputs. Abnormal situations like data overflows, communication failures, error handling, and recovery are not encountered. Unlike some programs, the exact sequence of operations is not important. The output generated is the customized

webpage.

#### 1.3.3 Performance Requirements

This script is designed to run on one computer at a time. The number of simultaneous users supported is limited by the bandwidth of the Web Texturizer server and is beyond the scope of this project.

#### 1.3.4 Logical Database Requirements

The Web Texturizer scripts have no database requirement.

#### 1.3.5 Design Constraints

In the design phase, two standards were used. The first is the Unified Modeling Language (UML). This modeling language is the graphical notation used to express designs. It was used to develop the Deployment Diagram, the Use Case Diagram, and the Class Diagrams.

The second is object-oriented software engineering methods. It uses five main methods: establish core requirements, develop a model of behavior, create the architecture, evolve the implementation, and maintenance. It also has four micro processes: identify the classes and objects needed, identify the semantics of these classes, identify the relationships among these classes, and specify the interfaces required.

### 1.3.6 Software System Attributes

1.3.6.1 Reliability. The reliability of this script was verified through extensive testing of all features. The script performed consistently throughout the testing phase.

In Phase one, the results from the equations used to calculate the vector and matrix calculations were compared. The conclusion was that the percentages should be on a 100% scale.

In Phase two, all the menus and buttons, the Graphic User Interface (GUI), were tested. Each selection from the menus performed its function as designed.

In the final phase of testing, the calculation of vector and matrix was added to the GUI to complete the application. Again, all functions and calculations performed without any failures.

1.3.6.2 Availability. This script is available anytime the Web Texturizer server is running. Presently, the server runs 24 hours a day and seven days a week.

1.3.6.3 Security. There is no security, such as passwords, required by this software. The idea is for open access to the Web Texturizer tool. It is the server's responsibility to provide the security needed by the Web



Texturizer programs. The script does not have access to the files on the server other than the ones needed for the application to run. No personal data is stored about the user, hence privacy is not a concern.

1.3.6.4 Maintainability. The script runs in the Perl 5.0 environment. If the version of Perl or its modules is updated, then the web-server that the Perl code resides on for this project will need to be restarted. There are no other maintenance requirements for this script.

1.3.6.5 Portability. Perl as a programming language was designed to run on most platform types. For the Internet user, the code will port to any computer with the proper web enabled browser. For the user, who downloads the code, they will need a copy of Perl version 5.0.

## CHAPTER TWO

### NUMERICAL ANALYSIS OF WEB TEXTURIZER EQUATIONS

#### 2.1 Web Texturizer Equations

The underlying representation of for text is based on the data model used by WHIRL: A Word-based Heterogeneous Information Representation Language (3). In this model WHIRL adopts a key tool of modern text-based information systems: the *term-weight* representation for text in which a document is represented as a set of *terms*, each associated with a numeric weight indicating its relative importance. Term-based representations can be easily created and stored, and with suitable indices many operations can be carried out. Hence, the term-weight representation creates a good weighing scheme (2). In WHIRL, the TF-IDF weighing scheme is used to analyze a document  $\bar{v}$  from some collection of documents  $C$ . In Web Texturizer this theory is applied to analyze a paragraph  $\bar{v}$  from a collection of paragraphs in a document  $C$ . It is an effective model for paragraphs with intuitively similar semantic content often have similar representations.

In the WHIRL data model, the items that are manipulated by the logic are not constant values, but

entities that correspond to fragments of text. We call these *simple texts*—"simple" emphasizing the text to have no additional structure. For example, in representing the information, simple text entities can be used to represent text fragments "Theater", "Space", "Harry", "Porter", etc. Hence, "simple texts" are a collection on keywords in a paragraph. Each simple text is represented internally as a *paragraph vector*. The paragraph vector consists of the terms that are components of the paragraph vector equal to the paragraph's characteristic stemmed keywords. We assume a vocabulary  $T$  of *terms*, that are word stems produced by Porter Stemming algorithm (6). The concept behind the vector representation is the magnitude of the component  $v^t$  is related to the importance of term  $t$  in the paragraph represented by  $\bar{v}$ . The TF-IDF weighing scheme is used. The term TF stands for Term Frequency. IDF stands for Inverse Document Frequency. Conceptually IDF is a weight for term frequency. Therefore, a term is weighted higher if fewer paragraphs in the entire document contain the term, making the terms more characteristic of the paragraph it appears in. Let  $\bar{v}$  represent a paragraph from a collection of paragraphs  $C$ . Hence,  $v^t$  is zero if the term  $t$  does not occur in the text represented by  $\bar{v}$ , else the equation is:

$$v^t = \log(\text{TF}\bar{v}_{,t} + 1) \cdot \log(\text{IDF}_t) \quad (\text{Equation 1})$$

In this formula,  $\text{TF}\bar{v}_{,t}$  is the number of times the term  $t$  occurs in the paragraph represented by  $\bar{v}$ , and

$$\text{IDF}_t = \|C\|/n_t \quad (\text{Equation 2})$$

where  $n_t$  is the total number of paragraphs in  $C$  that contain the term  $t$ . In Web Texturizer,  $C$  is the collection of paragraphs in a web page.

The advantage of this "vector space" representation is that the similarity of two paragraphs can be easily computed. The *similarity* of two paragraph vectors  $\bar{v}$  and  $\bar{w}$  is represented by the formula:

$$\text{SIM}(\bar{v}, \bar{w}) = \sum_{t \in T} \frac{v^t \cdot w^t}{\|\bar{v}\| \cdot \|\bar{w}\|} \quad (\text{Equation 3})$$

This is interpreted as the cosine of the angle between  $\bar{v}, \bar{w}$ . Notice that  $\text{SIM}(\bar{v}, \bar{w})$  is always between zero and one, and will be larger if two vectors share many "important" terms. This is referred as Cauchy-Swartz inequality [3].

In the tool we can judge the similarity of two paragraphs  $P1$  and  $P2$  in the collection of paragraphs  $C$  as follows:

[1] Remove stop words (such as prepositions, and, or, the, a, etc) and stem words (such as gone, going, goes stemmed to go) in the entire collection of paragraphs.

[2] A "term vector" is a vector where each term from [1] indexes one vector field. There is one term vector for each paragraph in the collection (e.g.,  $\overline{v_1}$ ,  $\overline{v_2}$  for each  $P_1$ ,  $P_2$ ). The vector field is 0 or 1 depending on whether the indexing term is present in the paragraph.

[3] From the 0, 1 vector representation in [2] compute vectors with TD-IDF values for each paragraph. As a result, 0s in the term vector from [2] remain 0, and 1s in the term vectors from [2] turn into a computed sum represented by the TF-IDF formula. Let  $\overline{v_1}$  be the TF-IDF vector for  $P_1$  and  $\overline{v_2}$  be the TD-IDF vector for  $P_2$ . Hence, in vectors  $\overline{v_1}$ ,  $\overline{v_2}$ , different terms are associated with different TF-IDF values. 0 implies that the term does not play a key role in the corresponding paragraphs. The larger the non-zero value is, the more important the term is to the underlying paragraph. In each vector, there is a characteristic set of relatively high-valued terms for the underlying paragraph.

[4] compute the similarity between P1 and P2 with the  $SIM(\bar{v}, \bar{w})$  formula provided in Equation 3.

#### Role of Keywords:

A user can enter keywords of personal interest, and a certain "keyword count". The document that is being viewed contains as many as these keywords as possible. And the tool displays the document with these highlighted keywords. Therefore, the words that are highlighted in the paragraphs are the keywords entered by the user (if entered by the user), along with the high TF-IDF valued terms within the paragraph. The tool displays up to "keyword count" many of these high-valued terms in each paragraph. Then each paragraph is likely to have a different set of such highlighted terms with more similar paragraphs sharing more of these terms. When the user presses the skip next button, the paragraphs most similar determined by [1] - [4] are brought into focus in descending order i.e. from high percentage to low percentage.

## CHAPTER THREE

### DESIGN

#### 3.1 Architecture (Component Diagram)

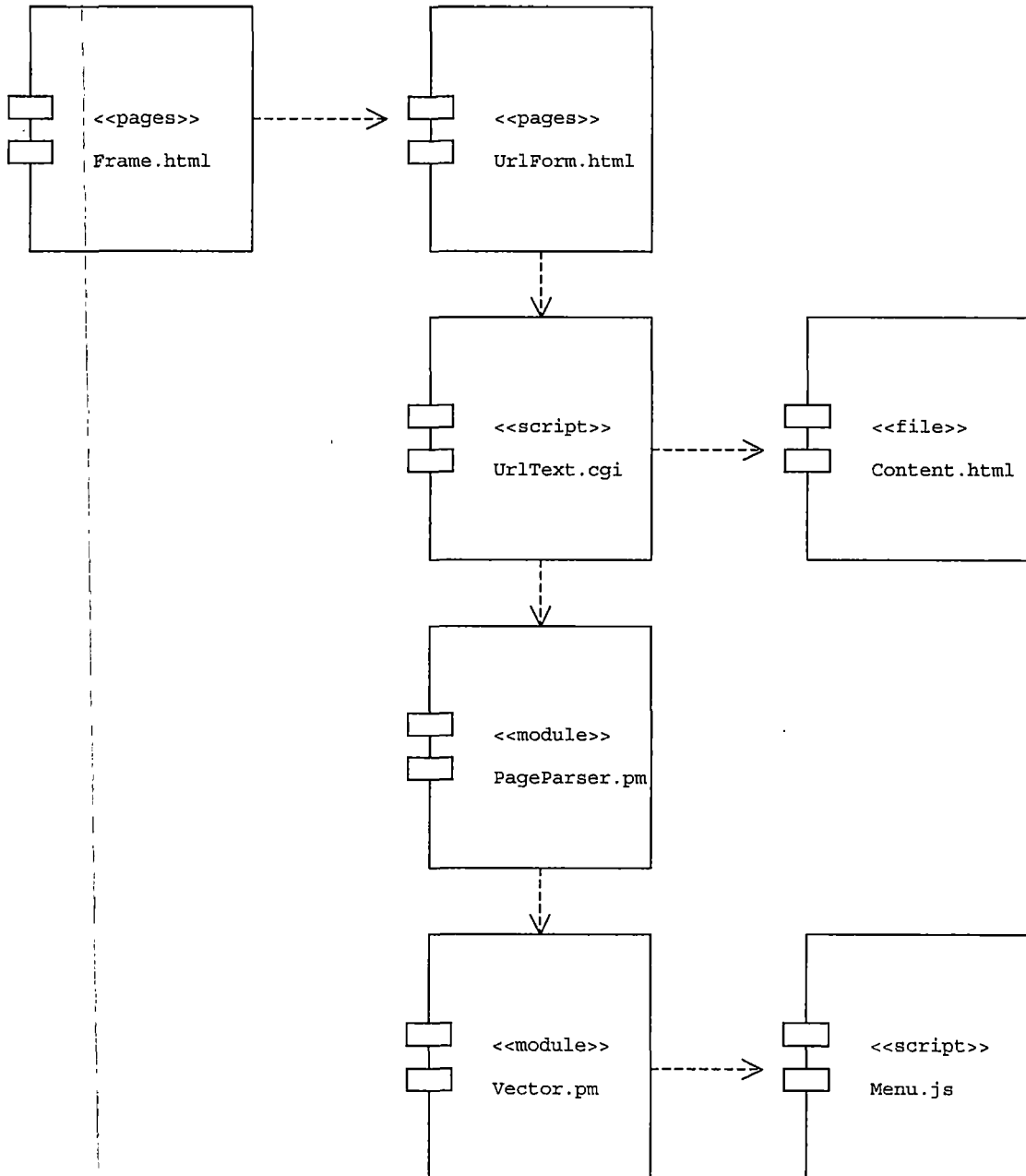


Figure 9. Component Diagram Overview

In Figure 9, the component diagram, an overview of the classes can be seen. In order to view the details of each of the component please see the class diagram of each of the components. Therefore, each class will be listed separately. In Table 2, UrlText class diagram is shown.

Table 2. UrlText.cgi Class Diagram

UrlText.cgi
url : string content : string
Init(cgi) getUrl() : String PageParser (content : String, url : String, stopwords : String, VFDDISPLAY : Boolean, MXDDISPLAY : Boolean)

Once the user enters the URL for the website to analyze, the UrlText.cgi script is started. It has the Init() function, which takes the place of the main() function and is executed first. This class has associations with the major classes of this project: PageParser.pm (Table 3), and Vector.pm (Table 4).



Table 3. PageParser.pm Class Diagram

PageParser
HTML KEYWORDS URL
Init (HTML : String, TEXT : String, VECTOR : String) dumpText2 (URL) references(html : String, href : String) hideLinks(par : String, htmlreferences : String, references : String) restoreLinks(par : String, htmlreferences : String, references : String) textAnalyze(text : String, stopwords : String, REDUCEDTEXT : Hash, STEMS : Hash, FATHOM : Hash, STEMCOUNT : Hash) dumpParseText(KEYWORDS : Hash, VECTOR : Hash, TEXT: Hash) fathomAnalyze(txt : String, FATHOM : Hash) dumpTopKeywords(num : String) dumpKeywords(uniq_words : Hash) getStemKeywords(list : Array, klist : Array) getStemParagraphs(plist : Array, klist : Array) getKeywords(list : Array) getParagraphs(list : Array)

The UrlText.cgi class creates a LWP connection to the website that is requested and gets the entire document. Light Weight Protocol (LWP) is a collection of modules in Perl. LWP creates network connections and manages the communication and transactions between client and server. The PageParser calls the Lingua module to stem the keywords based on Porter's Stemming algorithm [6]. It associates with Vector (Table 4), and Menu (Table 5).

Table 4. Vector.pm Class Diagram

Vector
VECTOR
Init (self : String, par : String, kys : String, Par : Hash, vectorKeys : Hash, V : Array, F : Array) Map(f : Array) Map(v : Array) matrix (I : String, String : j, MATRIX : String) mx(I : String, j : String) mxscript(ret : String) //returns par and vector SIM(VECTOR : String): string mxdisplay(I : String, j : String) //returns HTML matrix vfdisplay(MATRIX : Array, VECTOR : Array) //returns Sim and log calculations

The Vector is the class that maps the key and value pairs of keywords associated with paragraphs. It uses the similarity measures that are derived from William Cohen's paper called WHIRL: A word-based information representation language (2). It associates with Menu and the results are displayed on the content frame.

Table 5. Menu Class Diagram

Menu
Next : int Previous : int
nextpar(frm : String, dir : String) nextlink(frm : String, dir : String) setParNo(n : int) setParLink(n : int, m : int) //called when someone clicks on the parah on content frame setParValue(v : int, lnk : int)

The Menu class is JavaScript code that is used to set the default values that are calculated from the Vector class used in the Web Texturizer code. This class is designed to be used as user clicks through the next and previous buttons and scans through paragraphs with highest similarity measures. It has been added as a JavaScript engine to the Web Texturizer system.

### 3.2 Detailed Design (Pseudo-Code)

This section deals with the pseudo-code of the Web Texturizer. The tables are divided into two parts. The first part shows the overall scope of the class: class name, where used, purpose and note. The second part shows the pseudo-code.

For this project to run the CGI generated Perl script over the Internet, the browser firsts loads a HTML file from the Web Texturizer web site. The file, Urlform.html calls UrlText.cgi(Table 6).

Table 6. UrlText.cgi Pseudo-Code

Class Name	UrlText
Where Used	UrlForm.html
Purpose	Starts the Web Texturizer
Note	Web Texturizer Web Site, Main Page
Begin	Declare and Create Global Classes Get the URL using LWP Call PageParser.pm
End	

This calls and creates the classes used by the Web Texturizer perl script that runs on the web browser.

The module PageParser.pm (Table 7) is called by UrlText.cgi.

Table 7. PageParser.pm Pseudo-Code

Class Name	PageParser
Where Used	UrlText.cgi
Purpose	Stores the data for the 3 variables i.e. URL, keywords, and count
Note	Main frame of the program
Begin	Initialize the 3 variables Convert the HTML to ASCII Create dumpText2(URL) to convert HTML to ASCII Call as_Text() to convert HTML to ASCII text Create a list of keywords on a total basis Take the top N of these words as Keywords for each paragraph that contains 1 or more keywords, print list of keyword frequencies, and skip anchor which sends from 1 paragraph to next. And highlights keyword as link to the previous keyword Display the results
End	

PageParser calls Vector as a hash. Vector.pm described in Table 8 does the calculations based on the WHIRL system.

Table 8. Vector.pm Pseudo-Code

Class Name	Vector
Where Used	PageParser
Purpose	Calculates the similarity measure
Note	Calculates similarity measure between paragraphs
Begin Get the required parameters Calculate Vector keys Array Calculate the similarity measure vector Using the Array, find and display the matrix based on the calculations Using the Array, find and display the vector based on the calculations End	

Once the vector calculations are complete the results are returned to PageParser.pm. The results are displayed in content.html and the left frame gets an array of the matrix calculations. These are the key and value pair of the paragraph and its similarity measure. The results are also stored in an Array, which is used to navigate to 'next' and 'previous' paragraphs displayed by the next and previous buttons in Menu (Table 9).

Table 9. Menu Pseudo-Code

Class Name	Menu
Where Used	PageParser.pm
Purpose	Sets up the array of paragraphs and similarity measure for the Content Display area
Note	Left frame of the program
Begin Displays the array of the current paragraph Displays the array of the next similar paragraph Displays the similarity measure Results skip and scan on the content/display area. End	

## CHAPTER FOUR

### OPERATING INSTRUCTIONS

#### 4.1 Operating Instructions

The Web Texturizer has been designed for 2 groups of users- the novices and the experts. The main objective for both users is to reduce the "browsing effort". This objective is achieved by measuring keywords in text document as a vector.

The concept behind vector representation is the magnitude of a vector of keywords is related to the 'importance' of the keywords in the various paragraphs. Generally useful heuristics are firstly, to assign higher weights to terms that are 'frequent' in the document and secondly, to terms that are 'infrequent' in the collection as a whole.

In order to analyze the relevance between various paragraphs, three different websites were used. A research paper on computer games for children

(<http://www.terc.edu/mathequity/gw/html/MITpaper.html> ), a news story from an investment company website (

<http://flagship2.vanguard.com/web/planret/AdvicePTFinStartSetYourInvestmentPriorities.html>) and one concerning

Java programming language

(<http://java.sun.com/docs/white/langenv/Intro.doc2.html> ).

In the first research paper on "What Kinds of Educational Computer Games Would Girls Like?", the simple keywords that are of interest to a user are "games", "girls", and "computer". Using the Web Texturizer to browse through this document the paragraphs that are most related to one another are the ones that describe the reasons for these games, such as enhances "mathematical learning", a game called "Zoombinis" and "puzzle". A user can get the overview of the document by skimming the document and reach a conclusion that the games like Zoombinis help enhance the logical reasoning and mathematical skills.

The individualized browsing tool has two primary functions. First, the novice user that browses the document may have low expertise to the subject matter presented in the web page. Hence, the relevant information is presented coherently to the user by reducing the "browsing effort" (9). Second, an expert will have high expertise in the browsed subject matter. This tool will provide rapid access of personally relevant keywords to the expert user. (8)



The main function of the intelligent agent tool is the demonstrate methods for recognizing structure in HTML documents. Automating the process of finding keywords provides aids users to cope with the wealth of information available on the world wide web. Hence, the browser can highlight to the user potentially interesting but previously overlooked sources of information (10).

#### 4.2 Hints for the Users

This tool has been designed to allow the demonstration of notion on extracting structured information in web pages without any page-specific program training.

The main goal was to allow you, the user, to retrieve highly relevant information quickly and preserve coherence of information (11). The structure recognition methods were based on natural heuristics, such as detection of sequences of markup commands, and repeated patterns of "familiar-looking" strings. For example,  $C$  is a collection of simple text paragraphs that occur in Figure 10. The paragraph vector for text "Men in Black" in collection will be highly similar to vector  $v$ : as these simple paragraphs differ only on term "2002", which has high document frequency and hence low weight. The simple text paragraphs

for "Men In Black", "Stuart Little" and "Spiderman" will have low but non-zero similarity as they share "super" and "earth".

Movie	Review
Men In Black II, 2002	highly funded yet unofficial government agency that regulates all things alien on earth
Stuart Little II, 2002	the super intelligent mouse, and his adoptive human family
Spider Man, 2002	Battle with the super hero and villains of the Marvel earth
.....	

Figure 10. Table Containing Text

The selection of the Demo option displays the matrix and the vector calculation results. This way a user can visually review the results of the analyzed web document.

One of the limitations of this program is the inability to show the images i.e. the jpegs that maybe used in the document. The code is implemented to parse textual data. In addition, the web page may timeout due to restrictions set on the browser's domain. This prevents the document or a script loaded from one origin from

getting or setting properties of a document from another origin.

An advantage of the perl script is that it is a server side script. The server side scripts are friendlier on the visitor. The server is doing all the work and hence it does not matter what browser the visitor is using. This allows for transparent browser sniffing and customization.

### 4.3 Testing

The majority of the testing of this script was performed at <http://www.textseem.ehost4u.com/webtext-dev/>. There are no cases of a user being able to crash the script. The only input from the user is the mouse and a keyboard. This makes it almost impossible for the user to do anything that will crash the script. The script may timeout if there are restrictions on the documents domain.

### 4.4 How to Install Web Texturizer

For the users with some Unix and perl experience, it is possible to install and extend the perl modules. However, this will require all the files to be downloaded to the user's site and run there.

Step 1. From the Web Texturizer web site, download the file:

WebTexturizer.zip to the location on the user's host computer.

Step 2. Unzip the file in a folder this command.

Type:

```
gunzip WebTexturizer.zip
```

Step 3. Create a new folder in the location where all the htmls are located ex. public\_html.

Type:

```
mkdir webtext-dev
```

Step 4. Copy all the html files to this directory public\_html/webtext-dev. The files in this directory should be as follows-

```
index.html, menu.html, urlform.html, frame.html,  
content.html
```

Step 5. Copy all the images to a public\_html/webtext/images

Step 6. Using a text editor, such as vi or emacs open the urlform.html file. Using the search function, locate the three places where the word url\_text.pl is. At this location, insure that the line segment reads:

```
"/cgi-bin/webtext-dev/url_text.pl"
```

Of course if you put the files in another location, it will be your responsibility to change the location of these

three values to the proper path so the script can find the required files. Once the proper paths have been changed, save the file.

Step 7. Copy all the perl files to cgi-bin/webtext. The files in this folder are as follows: url\_text.pl, PageParser.pm, Vector.pm and stop\_words.txt

Step 8. Create a directory in cgi-bin/webtext-dev called "tmp". This is where all parsed keys and paragraphs are stored. The file names are keys999.txt and par999.txt are stored, where numbers '999' are random numbers that are generated by perl code.

Type:

```
mkdir tmp
```

Step 9. Test the results with IE browser by typing the URL where the site is hosted.

Type:

```
http://www.textseem.ehost4u.com/webtext-dev
```

Once in the IE browser, enter any website that you like to review and navigate through the site. Have fun!

## CHAPTER FIVE

### MAINTENANCE

#### 5.1 Files

The files for Web Texturizer are stored in the Web Texturizer Server Computer. In addition, all the files are copied on a CD ROM disk. A copy of this disk is stored in the back of the hardbound copy of this project stored in the Computer Science Office.

There are four main types of files used by this script: Perl source code (\*.pm), the images (\*.jpg), zip compressed files (\*.zip), and HTML web pages (\*.html). Inside the HTML files, several graphic styles are used, for example, jpeg, and gif.

The source files are displayed in Table 10. These source files created for this project are stored in ASCII format and can be viewed and modified by a text editor type program such as vi, emacs or notepad.

Table 10. Perl Modules that Need to be Installed

UrlText.pl	LWP- make html connection
PageParser.pm	Lingua - stem keywords
Vector.pm	

Table 11, shows the HTML files. These are the files used by the browser and are stored in ASCII format. They can be viewed and modified by a text editor.

Table 11. HTML Files

index.html
Frame.html
Urlform.html
Menu.html
Content.html

## 5.2 Directories

In the Web Texturizer Server Computer, there are several directories or folders used by the Web Texturizer project. The folder to find the main Web Texturizer directory is 'textseem'. Upon opening the 'webtext-dev' directory, several folders are displayed: 'cgi-bin' for the perl scripts, 'webtext-dev' for the html files, images to store the gifs. Opening these folders will display the files in which all the script files are stored. Therefore, the total path to this script files is:  
/usr/local/apache/public\_html or /usr/local/apache/cgi-bin.  
As Perl skilled users can download and modify this script, it was felt that all files would need to be in one location. This way a user would not have to download the

entire Web Texturizer system to be able to use the Web  
Texturizer demo at home.



## CHAPTER SIX

### WEB TEXTURIZER ANALYSIS

#### 6.1 Web Texturizer Versus Subjective Opinion

From probability and statistics, we know that the Web Texturizer could make two kinds of mistakes as described in Figure 11:

	Web Texturizer Results	
Subjective Opinion	Accepts paragraphs as related	Rejects paragraphs as unrelated
Paragraphs are related	<i>Correct decision</i>	<i>Type I error</i>
Paragraphs are unrelated	<i>Type II error</i>	<i>Correct decision</i>

Figure 11. Web Texturizer Versus Subjective Opinion

Correct Decision - Paragraphs are related in subjective opinion and Web Texturizer accepts paragraphs as related

In subjective opinion two paragraphs are related because they are talking about the same issues. Web Texturizer is a percentage based system. On seeing the same keywords being used to discuss the same issues it presents the correct result. It accepts the two paragraphs as related.

For example, review the following two paragraphs.

Paragraph 1: "Contribute as much as you can to employer-sponsored retirement plans. It's easy and convenient. You typically get a tax break on contributions—so putting a dollar toward retirement means giving up perhaps only 60 or 70 cents in spending power now. If your employer offers to match contributions, put in at least enough to get the entire match. If you don't, you're passing up a free pay raise."

Paragraph 2: "Everyone is eligible to contribute to an IRA. The only question is which type you should choose—a traditional IRA or a Roth IRA. The answer depends on your financial situation. A traditional IRA may or may not offer an up-front tax deduction on your contributions, depending on your income, but it always offers tax-deferred investment growth. A Roth IRA is never deductible, but it offers tax-free withdrawals during your retirement."

Result: On analysis with the Web Texturizer a 31% relationship was found based on the keywords shared between the paragraphs. This is a true result.

Correct Decision - Paragraphs are unrelated in subjective opinion and Web Texturizer rejects paragraphs as unrelated

In subjective opinion two paragraphs are unrelated because they are talking about the different issues. On seeing completely different keywords being used, the Web Texturizer rejects the two paragraphs as unrelated.

For example, review the following two paragraphs.

Paragraph 1: "Contribute as much as you can to employer-sponsored retirement plans. It's easy and convenient. You typically get a tax break on contributions—so putting a dollar toward retirement means giving up perhaps only 60 or 70 cents in spending power now. If your employer offers to match contributions, put in at least enough to get the entire match. If you don't, you're passing up a free pay raise."

Paragraph 2: "To address these issues, we are currently working on a project called Through the Glass Wall:

Computer Games for Mathematical Empowerment. Our research focus is on computer games that (potentially) teach mathematics; we are investigating the following questions: How do children learn significant mathematics from computer games? What are the characteristics of games, and of game-playing contexts that interact with learning? What patterns

are there in girls' and boys' approaches to games -- and in their learning from these games? "

Result: On analysis with the Web Texturizer 0% relationship was found based on the keywords shared between the paragraphs. This is a true result.

Type I error - Rejecting the two paragraphs as unrelated, when they are related

Users have more knowledge about the subject being talked about, which is inherent in their mind due to the experiences or the understanding of the subject from the previously read material. Also, users have a large vocabulary to choose from and when they are writing they may use few different ways to talk about the same subject and may assume that the reader would understand what they are saying from the context of the subject. Hence, this is an advantage over an algorithm that is purely keyword based to find key relationships between two paragraphs.

For example, if there were two paragraphs to be analyzed talking about the exact same subject and in paragraph 2 the synonyms of all the keywords from paragraph 1 are used. This keyword based algorithm would not be very successful in ascertaining the degree of relationship.

Following are the two paragraphs that were used to detect a correct relationship by the Web Texturizer in case 1 above (=31%). The keywords were replaced in Paragraph 2 with their synonyms (retirement, offer, contribute, tax). The two paragraphs were again analyzed with the Web Texturizer.

Paragraph 1: "Contribute as much as you can to employer-sponsored retirement plans. It's easy and convenient. You typically get a tax break on contributions—so putting a dollar toward retirement means giving up perhaps only 60 or 70 cents in spending power now. If your employer offers to match contributions, put in at least enough to get the entire match. If you don't, you're passing up a free pay raise."

Paragraph 2: "Everyone is eligible to add to an IRA. The only question is which type you should choose—a traditional IRA or a Roth IRA. The answer depends on your financial situation. A traditional IRA may or may not present an up-front deduction on your donations, depending on your income, but it always presents tax-deferred investment growth. A Roth IRA is never deductible, but it presents tax-free withdrawals when you are over 65."

Result: On analysis with the Web Texturizer only a 2% relationship was found based on the keywords shared between the paragraphs. This is a false result.

Type II error - Accepting two paragraphs as related, when they are unrelated. Let's take the opposite of Type I error case. The same exact words may be used to describe two different unrelated subject matters. The keyword algorithm based system would make a mistake in identifying those two paragraphs as related.

Following are the two paragraphs that were used to detect an incorrect relationship by the Web Texturizer. The keywords are the same in the two paragraphs (software, teens, and girls). The two paragraphs talk about unrelated subjects. The first paragraph talks about marketing goods to female teens online. The second paragraph relates to gaming software for girls and boys.

Paragraph 1: "While the online teen population is expected to continue to increase, female teens will still spend a limited amount of time online. Businesses that target the teen girls market must evaluate their content and offer elements that these teens want in order to capture any part of that limited time," said Anya Sacharow, Jupiter's analyst for teen girls markets. "Girls follow offline

brands online, but boys just want what they are looking for and don't seem to care where it comes from. Strong brand building and alliances with online networks sway teen girls; teen boys are software experts largely."

Paragraph 2: "One backdrop to our research is the current condition of commercial game software. Girls have only recently been identified by the computer industry as a potentially profitable market. The last few years have seen the emergence of a new set of software targeted particularly at girls. Much of this software appeals to stereotypically "female" interests: shopping, make-up, fashion, dating. Some have even called this category of software "pink." Our initial survey of the games market revealed (perhaps not surprisingly) that there were no games aimed specifically at girls that required significant mathematical thinking."

Result: On analysis with the Web Texturizer a 29% relationship was found based on the keywords shared between paragraphs. This is a false result.

Recommendations:

- 1) The keyword based Web Texturizer should be able to decipher synonyms and hence use them to compute the measure for relationships between paragraphs.

2) Qualitative or subjective analysis needs to be imparted to the software by use of more sophisticated algorithms.



## CHAPTER SEVEN

### FUTURE DEVELOPMENTS AND CONCLUSIONS

#### 7.1 Ideas for Future Developments

There are a few limitations to the program as written. The first is there is no way to update the contents of browser options as personal relevance profiles changes over time. At present, the user has to enter the web page every time it needs to be analyzed. This is a partial fix for this problem. If a Perl skilled user downloads the files, they would be able to change the code to build and maintain for each browser user a profile that records those documents and keywords that are deemed relevant to the user at the time of the next browser session.

A more advanced version of the Web Texturizer will attempt to infer relevance of unviewed keywords from various form of relatedness to relevant previously viewed keywords in a document as the users expertise changes in the browsed subject.

The computing world is always changing. Web Texturizer can be updated and re-written in Python. Perl acquired a strong ecological niche in the early 1990s as a CGI programming language, it's familiar to people who have a

strong Unix background, and its convenience for text processing is one of Perl's strongest points. But the computing world is always changing: many people who write scripts may now have a background in Windows development, or in Java, or no programming experience at all. In the years to come, increasing numbers of people will be attracted to Python's conceptually simpler organization and its shallower learning curve. Python is an excellent tool for scanning and manipulating textual data. Python is a freely available, very high-level, interpreted language developed by Guido van Rossum. It combines a clear syntax with powerful (but optional) object-oriented semantics. Python is widely available and highly portable.

## 7.2 Conclusions

All the design features proposed for this project have been implemented using consistent naming conventions to name the parameters, variables, and perl modules. The script runs on the Web Texturizer web site; it accurately demonstrates WHIRL system; and it allows users to quickly navigate through the web page. The program does all that was hoped for in the initial design phase.

The benefits of this script are easily demonstrated. For novices, it allows them speedier access to relevant information instead of reading the entire document. For experts, it has speedier retrieval of relevant information.

APPENDIX A  
WEB TEXTURIZER CODE

## PAGEPARSER PERL CODE

### PageParser.pm

```
package PageParser;
use strict;
use FindBin qw($Bin);

# Simple extension of Treebuilder to parse through a file,
and add
# in the capability to skip through the text

use Lingua::EN::Fathom;
use HTML::TreeBuilder;
use HTML::Element;
use HTML::FormatText;
use URI::URL;

use Vector;
use Lingua::Stem;
use Lingua::Ispell qw( spellcheck );

my %defaults = (
    HTML => undef, # The HTML as a text string
    KEYWORDS => undef, # Keywords as a hash, count of
keys in analyze
    URL => undef, # The path to wherever
);

sub new {
    my $class = shift;

    my %extra;
    my $self = {};
    bless($self, $class);
    @$self{keys %defaults} = values %defaults ;
    if(@_) {
        my %stuff = @_;
        @$self{keys %stuff} = values %stuff;
    }
    $self->init();
    return $self;
}

sub init {
```

```

my $self = shift;

if($self->{HTML}) {
    $self->{TEXT} = $self->as_Text();
    $self->txtAnalyze();
    $self->{VECTOR} = Vector->new( $self-
>getStemParagraphs() );
}
}

sub as_Text {
    my $self = shift;
    return($self->dumpText2());
}

# Use lynx to get it into a more reasonable ascii format
sub dumpText2() {
    my $self = shift;
    my $ret = "";
    my $url = $self->{URL};
    my @text = split /\r?\n/, `lynx -dump -width=4000
$url`;
    # Pass 1 -- clear out blank lines, [image] lines, and
    # meaningless junk
    my @text1 = map {
        my $line = $_;
        # remove long meaningless --- and ____ strings
        $line =~ s/[!@#%&*()\-_+=][!@#%&*()\-_+=]+//;
        if( $line =~ m/^\s*\[.*\]\s*$/ ) {
            $line = undef;
        }
        $line
    } @text;
    # Pass 2 -- Look for short lines that are probably
header lines to
    # paragraphs. Combine those into the next paragraph.
Look for
    # long lines that may have been broken up. Should
have a 7 space
    # indent on them. Join those lines together.
    my $line;
    while (@text1) {
        $line = shift(@text1);
        if($line =~ m/^\s*$/ ) {next; }
        if( $line =~ m/^\s*References\s*$/ ) {

```

```

        $self->references(\@text1);
        last;
    }
    # Arbitrarily assume that a header line is less
    than 50 chars
    # $words_num = $#line;
    if(length($line) < 50 ) {
        $line = "<h>$line</h>\n";
        my $line2 = shift(@text1);
        if($line2) {
            if(length($line2) > 50) {
                $line .= "\n$line2";
            }
            else {
                unshift @text1, $line2;
            }
        }
        else {
            unshift @text1, $line2;
        }
    }
    while(length($line) > 60) {
        my $line2 = shift(@text1);
        if(!$line2) {
            last;
        }
        if($line2 =~ m/^          \w+/) {
            $line .= $line2;
        }
        else {
            unshift @text1, $line2;
            last;
        }
    }
    # trim off useless space
    $line =~ s/^\s*(.*)\s*$/\1/;
    $ret .= "<P>\n$line\n</P>\n";
}
return $ret;
}
# This get the lynx reference set
# found at the end of the lynx output
sub references {
    my $self = shift;
    my $text = shift;

```

```

my %ref = ();

for( @$text) {
    if( m@(\d+)\.s*([^\s#]+\s*@ ) {
        my $num = $1;
        my $rf = $2;
        $rf =~ s@/$@@;
        if($rf !~ m/mailto/i) {
            $ref{$num} = $rf;
        }
    }
}

$self->{REFERENCES} = \%ref;
# Now go and get the interesting references from the html
my $html = $self->{HTML};
my @str = ();
my %hstr = ();
$html =~ s{<A\s.*?HREF="?([^\s">]*)[^\s">]*?>(.*)}{ push
@str, ($1, $2); "$&" }iges;
while(scalar(@str)) {
    my $href = shift @str;
    my $val = shift @str;
    $href = url($href, url($self->{URL}))->abs-
>as_string;
    $href =~ s@/$@@;
    $val =~ s/<[^\s">]*>/ /gs;
    $val =~ s/^\s*(.*)\s*$/$1/;
    $hstr{$href} = $val if $val ;
}
$self->{HTMLREFERENCES} = \%hstr;
}

# Pass through the paragraph, reconstructing links that
were in the original

# HTML except now they go through the Javascript stuff.

sub hideLinks {
    my $self = shift;
    my $par = shift;
    my $str = $self->{HTMLREFERENCES};
    my $rf = $self->{REFERENCES};

    return $par unless $rf;
    return $par unless $str;
    my @num = ( $par =~ m/\[(\d+)\]/gs );

```



```

        for (@num) {
            if($rfr->{$_}) {
                my $re = "\\[$_\\]\\s*".$str->{$rfr->{$_}}.'\s*';
                unless ($par =~ s{$re}{\\[$_]}) {
                    $par =~ s/\\[$_]//;
                }
            }
        }
        return $par;
    }
}
# Recover the links that lynx inconveniently put at the end
of the file
# and put them in their rightful place in the document.
sub restoreLinks {
    my $self = shift;
    my $par = shift;
    my $str = $self->{HTMLREFERENCES};
    my $rfr = $self->{REFERENCES};

    return $par unless $rfr;
    return $par unless $str;

    my @num = ( $par =~ m/\\[(\\d+)\\]/gs );

    for (@num) {
        if($rfr->{$_}) {
            my $re = "\\[$_\\]\\s*";
            $par =~ s{$re}["<A
HREF=\"javascript:linkto('".$rfr->{$_}.\"')\">".$str->{$rfr->{$_}}.\"</A>"]gei;
        }
    }
    return $par;
}

```

```

#####
# The following methods are adaptation from url_fathom
# or my interpretation of what should happen
# txt_analyze --
# 1. create a text version of the html
# 2. strip it of "stop words", and send to fathomAnalyze
# 3. create a list of the words on a total basis
# 4. take the top N of these words as Keywords
# 5. create a html document which analyzes each paragraph

```

```
# highlights each keyword as a link to previous keyword
# for each paragraph that contains 1 or more keywords,
# print a list of the keyword frequencies, and a skip
# anchor which sends it from one paragraph to the next.
# Strip "stop_words" Are these common prepositions, etc.
```

```
sub txtAnalyze {
    my $self = shift;
    my $text = $self->{TEXT};
    my $key1_splited = $self->{EXTRAKEY};
    my @key1_splited;
    my $tmp;
    my $var = ref($key1_splited);
    my $size = scalar(@{$key1_splited});
    my $stopwords = $self->{STOPWORDS};
    unless( $stopwords ){
        $stopwords = $self->stop_words();
        $self->{STOPWORDS} = $stopwords;
    }
    # remove all the stopwords
    my $re = "(\\b" . join ("\\b|\\b", @$stopwords) .
"\\b)";
    my $txt = $text;
    $txt =~ s/$re//gis;
    $self->{REDUCEDTEXT} = $txt;
    # Fathom analyze to get keywords
    my $keywords = $self->fathomAnalyze($txt);
    # nip the rest down to their stems ...
    my $stemmer = Lingua::Stem->new( -locale => 'EN');
    $stemmer->stem_caching({-level => 1});
    my $stems = $stemmer->stem(@$keywords);
    my %stemhash = ();
    for (0..$#$keywords) {
        my $tmpstem = $stems->[$_];
        #      print "TMP Stems is : $tmpstem<br> ";
        $stemhash{$keywords->[$_]} = $stems->[$_];
    }
    $self->{STEMS} = \%stemhash;

    # Now, do a re-count based on stemmed words
    my $fathom = $self->{FATHOM};
    my %uniq_words = $fathom->unique_words;
    my %keycount;
```

```

    for (keys %uniq_words) {
        my $tmp1 = $uniq_words{$_};
        my $tmp2 = $stemhash{$_};
        $keycount{$stemhash{$_}} += $uniq_words{$_};
    }
    $self->{STEMCOUNT} = \%keycount;
    # Now, get the top 10 keywords
    ($self->{STEMKEYWORDS}, $self->{KEYWORDS}) = $self->
>getStemKeywords(10);
# dump out the parsed text, putting in some html tags to
make the
# display more useful.
sub dumpParseText {
    my $self = shift;
    my $keywords = $self->{KEYWORDS};
    my $ret = "";
    my $vector = $self->{VECTOR};
    # Create the HTML document
    my $re = "(\\b" . join("\\b|\\b", @$keywords) .
"\\b)";
    my %kws = ();
    map { $kws{$_} = 0; } @$keywords;
    $ret = $self->dumpTopKeywords();
    my $txt = $self->{TEXT};
    my $parno = 0; # paragraph number
    my $lastparno = 0;
    my $curpar = 1; # was 0
    my @klist = ();
    # Scan by paragraph to set the links
    $ret .= "<TABLE cellpadding=0 cellspacing=0>\n";
    while( $txt =~ m@<P>\n(.*)\n</P>@gis ) {
        my $par = $self->hideLinks($1);
        my %kw = ();
        #
        if( $par =~ s{$re}[ $kw{lc($&)} = 1; "<A HREF=#"
. lc($&) $kws{lc($&)}++ " NAME=" . lc($&) . $kws{lc($&)} .
" >$&</A>" ]gei ) {
            $ret .= "<TR class=Bg1><TD><A NAME=p$curpar
HREF=\"javascript:pswitch($curpar)\">>$curpar.</A> ";
            if( $par =~ s{$re}[ $kw{lc($&)} = 1;
"<B>$&</B>" ]gei ) {
                # Wow, there are keywords in this paragraph!
                push @klist, ("\"par$curpar\"");
                $parno += 1;
            }
        }
    }
    ### insert header check here

```

```

        if ($par =~ m/^\<h>.*\</h>/)
        {
            $ret .= "<A NAME=par$curpar>KEYWORDS-DEV: "
. join(" ", keys %kw) . "</A>";
        }
        $ret .= "</TD></TR>\n";
        $par = $self->restoreLinks($par);
        $par =~ s/\n/<br>\n/gis;
        $ret .= "<TR class=Bg2><TD>" . $par . "</TD></TR>\n";
        $curpar += 1;
    }
    $ret .= "</TABLE>\n";
    if($self->{MXDISPLAY}) {
        $ret .= "<P>" . $vector->mxdisplay();
    }
    if($self->{VFDISPLAY}) {
        $ret .= "<P>" . $vector->vfdisplay();
    }
}
my $js = <<"EOT";
<STYLE>
BODY
{
    BACKGROUND-COLOR: white;
    FONT-FAMILY: Verdana, Arial, Helvetica;
    FONT-SIZE: 8pt;
    MARGIN: 4px
}
TD
{
    FONT-FAMILY: Verdana, Arial, Helvetica;
    FONT-SIZE: 8pt
}
.Bg1
{
    BACKGROUND-COLOR: #dddddd
}
.Bg2
{
    # BACKGROUND-COLOR: #eeeecc;
    # FONT-SIZE: 10pt;
    BACKGROUND-COLOR: #dddddd;
    FONT-SIZE: 8pt;
}
.Bg3
{

```

```

        BACKGROUND-COLOR: #dddddd;
        FONT-SIZE: 8pt;
    }
</STYLE>
<script language=javascript>
EOT
$js .= "var klist = new Array(" . join( ",", (@klist)) .
");\n";
$js .= <<"EOT";
parent.menu.setParNo(klist.length);
function kvalue(n) {
    return klist[n];
}
EOT
$js .= $vector->mxscript();
$js .= <<"EOT";
function pswitch(par) {
    // alert("pswitch -- "+par);
    parent.menu.setParLink(par, plist[par].length);
}
function setLocation(par, n) {
    // alert("setLocation -- "+par+" "+n);
    // alert("Hash set " + plist[par][n]);
    parent.menu.setParValue(vlist[par][n], plist[par][n]);
    // document.hash.location = '#' + plist[par][n];
    // alert("Hash Location -- "+document.hash.location);
}
function linkto(url) {
    parent.nav.newurl(url);
}
</script>
EOT
return ($js , $ret);
}
# Read in the stopwords.
sub stop_words {
    my $self = shift;
    open KW, 'stop_words.txt' or die $!;
    my @kw = map {chop;$_} <KW>;
    close KW;
    return \@kw;
}
# CREATE A BEST WORDS LIST

sub fathomAnalyze {

```

```

my $self = shift;
my $txt = shift;
my $fathom = $self->{FATHOM};
if(!$fathom) {
    $fathom = new Lingua::EN::Fathom;
    $self->{FATHOM} = $fathom;
    $txt =~ s@</?P>@@gis;
    $fathom->analyse_block($txt);
}
my %uniq_words = $fathom->unique_words();

    my @kws = keys %uniq_words;
##print "KWS new : @kws <BR>";
    return \@kws;
}
# dump the top 10 keywords
sub dumpTopKeywords {
    my $self = shift;
    my $count = $self->{COUNT}; ## Count of the number
of keywords to analyze
    return $self->dumpKeywords('num', $count);
}
sub dumpKeywords {
    my $self = shift;
    my $dir = shift; # sort by either alpha, or num
    $dir = "alpha" unless $dir;
    my $len = shift;
    $len = 0 unless $len;
    my %uniq_words = %{$self->{STEMCOUNT}};
    my $word;
    my $ret;
    my @list = sort keys %uniq_words;
@list = sort keys %uniq_words;
    if($dir eq 'num') {
        @list = sort { $uniq_words{$b} <=>
$uniq_words{$a} } keys %uniq_words;
    }
    if($len) {
        splice @list, $len;
    }
    $ret = "<TABLE>\n";
    foreach $word ( @list )
    {

```

```

        $ret .= "<TR><TD ALIGN=right>" .
$uniq_words{$word}. "</TD><TD>$word</TD></TR>\n"; # outputs
the word and frequency.
    }
    $ret .= "</TABLE>\n";
    return $ret;
}
# Get the top n Stem Keywords. Also generate the
equivalent array
# of real keywords (which will have more than n keys, and
display unstemmed)
sub getStemKeywords {
    my $self = shift;
    my $len = shift;
    my $stems = $self->{STEMS};
    my $stemcount = $self->{STEMCOUNT};
    my $key1_splited = $self->{EXTRAKEY};
    my $ tmp;
    my $textk = $self->{TEXT}; ## Text that contains the
keywords
    my @list = sort { $stemcount->{$b} <=> $stemcount->
{$a} } keys %$stemcount;
    splice @list, $len;
    my @key1_splited;
    # now find all the words in the other list
    my @klist = ();
    foreach $tmp (@{$key1_splited}){
        push (@klist, $tmp);
    }
    print "Extra keywords: <br>";
    foreach $tmp (@{$key1_splited}){
        my $count = () = $textk =~ /\b\Q$tmp\E\b/gi;
        print "$count    $tmp    <br>";
    }

    for (keys %$stems) {
        my $w = $_;
        for (@list) {
            if($stems->{$w} eq $_) {
                push @klist, $w;
                last;
            }
        }
    }
    return( \@list, \@klist );
}

```

```

# replace words with stemmed words in the paragraphs.
sub getStemParagraphs {
    my $self = shift;
    my $text = $self->{REDUCEDTEXT};
    # Get list of multi-word stems
    my $stems = $self->{STEMS};
    my %seen = ();
    map { $seen{$_} ++ } values %$stems;
    my @list = grep { $seen{$stems->{$_}} > 1 } keys
%$stems;
    # replace those words in the text
    my $re = "(\b" . join("\b|\b", (@list) ) . "\b)";
    $text =~ s{$re}[ $stems->{lc($&)} ]gise;
    # break into paragraphs
    my @plist = ( $text =~ m@<P>\n(.*)\n</P>\n@gis );
    # create a key list
    my %kseen = ();
    my @klist = grep { ! $kseen{$_} ++ } map {
$seen{$stems->{$_}} > 1 ? $stems->{$_} : $_ } keys
%$stems;
    return (\@plist, \@klist);
}

sub getKeywords {
    my $self = shift;
    my $fathom = $self->{FATHOM};
    my %uniq_words = $fathom->unique_words;
    # my @list = grep { $uniq_words{$_} > 1 } keys
%uniq_words;
    my @list = keys %uniq_words;
    return \@list;
}

sub getParagraphs {
    my $self = shift;
    my $text = $self->{TEXT};
    my @list = ( $text =~ m@<P>\n(.*)\n</P>\n@gis );
    return \@list;
}

1;

```



# VECTOR PERL CODE

## Vector.pm

```

package Vector;
use strict;
# Make the vector.pl into a package
# Input arguments to Vector are
# 1. a reference to an array of paragraphs
# 2. a reference to an array of keywords
sub new {
    my $class = shift;
    my $self = {};
    my @args = @_;

    bless($self, $class);

    $self->init(@_);
    return $self;
}
sub init {
    my $self = shift;
    my $par = shift;
    my $kys = shift;
    my @P = @$par;
    my @vector = @$kys;
    unshift @P, join"\n", @P;
    my @f = map{
    my $p = $_;
    my %f;
    @f{@vector} = map{ 0+@{[$p=~/\b$_/gi]} } @vector;
    \%f} @P;
    my @v = map{
        my $f = $_;
        my %v;
        @v{@vector} = map{
        ##log(1+$f->{$_})*log(3/$f[0]{$_})
        $f[0]{$_} && log(1+$f->{$_})*log($#P/$f[0]{$_})
        } @vector;
        \%v;
    }@f;
    $self->{VECTOR} = \@vector;
    $self->{V} = \@v;
    $self->{F} = \@f;
    # Create the matrix

```

```

my %matrix = ();
my ($i,$j);
for $i ( 1..$#v ){ for $j ( 1..$#v ){
    if($i == $j) {
        $matrix{"$i,$j"} = 1;
    } else {
        $matrix{"$i,$j"}=sqrt($self->SIM($v[$i],$v[$j]));
    }
} }
$self->{MATRIX} = \%matrix;
}
sub mx {
    my $self = shift;
    my $i = shift;
    my $j = shift;
    my $matrix = $self->{MATRIX};

    return $matrix->{"$i,$j"} ;
}
# Create a piece of javascript to recreate this matrix on
the browser
sub mxscript {
    my $self = shift;
    my ($i, $j) = (0,0);
    my $matrix = $self->{MATRIX};
    my $v = $self->{V};
    my @vvv = @$v;
    my $ret = "";

    $ret = "var plist = new Array();\n";
    $ret .= "var vlist = new Array();\n";
    for $i (1..$#vvv) {
        ## my $ii = $i-1;
        my $ii = $i;
        my $pp = "plist[$ii] = new Array(";
        my $vv = "vlist[$ii] = new Array(";
        my %r = ();
        for $j (1..$#vvv) {
            ## my $jj = $j-1;
            my $jj = $j;
            if($matrix->{"$i,$j"}) {
                $r{$jj} = sprintf("%0.6f", $matrix-
>{"$i,$j"}));
            }
        }
    }
}

```

```

        # sort this stuff numerically
        my @app = ();
        my @avv = ();
        map {
            push @app, "\"p$_\"";
            push @avv, $r{$_};
        } sort { $r{$b} <=> $r{$a} } keys %r;
        $pp .= join ",", (@app);
        $vv .= join ",", (@avv);
        $ret .= $pp . ");\n";
        $ret .= $vv . ");\n";
    }
    return $ret;
}

# Note: This routine is only used internally
sub SIM{
    my $self = shift;
    my $vector = $self->{VECTOR};
    my($v1,$v2) = @_ ;
    my ($s0,$s1,$s2)=(0,0,0);
    for( @$vector ){
        $s1 += $v1->{$_}**2;
        $s2 += $v2->{$_}**2;
        $s0 += $v1->{$_}* $v2->{$_};
    }
    if($s1 == 0 || $s2 == 0) {
        return(0);
    }
    return $s0*$s0/($s1*$s2);
}

# Make a table display
sub mxdisplay {
    my $self = shift;
    my ($i, $j) = (0,0);
    my $matrix = $self->{MATRIX};
    my $vvv = $self->{V};
    my @v = @$vvv;
    my $ret = "";

    $ret = "<table cellpadding=1 cellspacing=1
border=0>\n";
    for $i (1..$#v ){
        ## my $ii = $i -1;
        my $ii = $i; # start at p1

```

```

    $ret .= "<tr class=bg3 ><td><a
href=\"#p$ii\">p$ii</a></td>";
    for $j (1..$#v ) {
        # $ret .= sprintf("<td>%5.2f</td>", $matrix-
>{"$i,$j"})
        my $m = $matrix->{"$i,$j"};
        my $c = qw("red" "orange"
"brown") [ ($m>0.3)+($m>0.6)+($m>0.8) ];
        $ret .= sprintf("<td><font
color=$c>%5.2f</font></td>", $matrix->{"$i,$j"})
    }
    $ret .= "</tr>\n";
    $ret .= "</table>\n";
    return $ret;
}

sub vfdisplay {
    my $self = shift;
    my ($i, $j) = (0,0);
    my $matrix = $self->{MATRIX};
    my $vvv = $self->{V};
    my @v = @$vvv;
    my $fff = $self->{F};
    my @f = @$fff;
    my @vector = @{$self->{VECTOR}};
    my $ret = "";
    $" = "<td>";
    $ret = "<table cellpadding=1 cellspacing=1
border=0>\n";
    $ret .= "<tr
class=bg3><td></td><td>@vector</td></tr>\n";
    for( 1..$#v ) {
        $ret .= "<tr
class=bg3><td>p$_</td><td>{@f[$_]}{@vector}</td></tr>\n";
    }
    $ret .= "</table>\n";
    $" = ",";
    $ret .= "<P>\n";
    for( 1..$#v ) {
        $ret .= "v$_=( {@v[$_]}{@vector} )<br>\n";
    }
    return $ret;
}
1;

```

```

END
$" = ",\t";
print "\t@vector\n";
for( 1..$#P ){
    print "p$_:\t@{$f[$_] }{@vector}\n";
}
$" = ",";
for( 1..$#v ){
    print "v$_=(@{$v[$_] }{@vector}))\n";
}
for $i ( 1..$#v ){ for $j ( 1..$#v ){
    print "SIM(v$i,v$j)=",sqrt(SIM($v[$i],$v[$j])), "\n";
}
}
#####MATIX FORMAT
for $i (1..$#v ){
    for $j (1){
        if( $i==$j ){
            $s[$i] = -1;
        }else{
            $s[$i] = SIM($v[$i],$v[$j]);
        }
    }
}
for $i (sort{$s[$b]<=>$s[$a]}1..$#v ){
    print "P$i";
    for $j (1..$#v ){
        if( $i==$j ){
            print "  x  "
        }else{
            printf("%5.2f",sqrt(SIM($v[$i],$v[$j])))
        }
    }
    print "\n";
}
}

```

## URL\_TEXT CODE

url\_text.pl

```
#!/usr/bin/perl -w
##-I/web/public/grad/sdesar
#####
#####
#This script does the following
#1. file.txt- converts html to ascii
#2. file.html- fetch the html file.
#3. fileParse.txt - Parse the articles, prepositions, etc.
to create the keywords to be analyzed.
#4. fileHeader.html- gets the headers from file.html and
creates anchors.
#5. stop_words- List of words which are to be  parsed....
a, the, and, etc.,
# I got this from http://www.nzdl.org
#This file contains the following Routines
#1. convert to plain_text.
#2. copy HTML file
#3. parse data
#4. get all the headers and place it after the <BODY> tag
and make them anchors
#####
use FindBin qw($Bin);
use LWP::Simple;
use HTML::Parser;
use CGI;
use PageParser;
my $cgi = new CGI;
my $url = $cgi->param('url');
my $key1 = $cgi->param('keywords');
my $count = $cgi->param('count');
my $type = $cgi->param('type');
$type = 'text' unless $type;
use CGI qw(:standard);
print $cgi->header(); # ** leave this to display keywords
my @key1_splited=split(",", $key1);#split to array of 3words
my $BASE = "$Bin/tmp";
my $body = "NOTHING";
my $js = ""; # Javascript to insert into the header;
my $STOPWORDS = [ "the", "a", "it", "and",
                  "some", "my", "your", "an", "did",
                  "his", "hers", "he", "him", "her",
```

```

    "she",      "that",    "those",   "then",    "was",
    "is",       "would",   "for",     "with",    "there",
    "of",       "such",    "they",    "by",      "you",
    "to",       "on",      "are",     "do",      "will",
    "in",       "be",      "how",     "can",     "our",
    "or",       "this",    "what",    "who",     "when",
    "we"];
if($url) {
    my $content = get( $url );
    if($content) {
        my $pp = PageParser->new(
            HTML => $content,
            URL => $url,
            STOPWORDS => undef,
            EXTRAKEY => \@key1_splited,
            COUNT => $count,
            VFDISPLAY => "", # Set to true if you want
            MXDISPLAY => "", # Set to true if you want);
# Statsh away some text files for testing purposes
        open OUT, ">$Bin/tmp/par$$$.txt";
        print OUT join( "\n\n", @{$pp->getParagraphs()} );
        close OUT;
        open OUT, ">$Bin/tmp/keys$$$.txt";
        print OUT join( "\n", @{$pp->getKeywords()} );
        close(OUT);
        # $pp->analyze();
        if($type eq 'text') {
            my $text = "";
            ($js, $text) = $pp->dumpParseText();
            $body = <<"EOT";
<BODY>$text</BODY>
EOT
            }
            elsif($type eq 'html') {
                $body = $pp->dumpHTML();
            }
            else { $body = <<"EOT";
<BODY><CENTER>
Content not found at $url.
</CENTER></BODY>
EOT
            }
        }else { $body = <<"EOT";
<BODY><CENTER>Please enter a URL.</CENTER></BODY>EOT}
print $cgi->header( -type => 'text/html' );
print <<"EOT";
<HTML><HEAD>$js</HEAD>$body</HTML>EOT
1;

```

## FRAME CODE

frame.html

```
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1252">
<title>Web Texturizer</title>
</head>

<frameset rows="70,*" cols="*">
  <!--
    <frame name="nav" scrolling="no" noresize
target="contents" src="test.cgi">
    -->
    <frame name="nav" scrolling="no" target="contents"
src="urlform.html">
    <frameset cols="190,*">
      <frame name="menu" target="main" src="menu.html">
      <frame name="content" src="content.html">
    </frameset>
  <noframes>
    <body>
      <p>This page uses frames, but your browser doesn't
support them.</p>
    </body>
  </noframes> </frameset>

</html>
```



## URLFORM CODE

urlform.html

```
<html>
<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1252">
<title>Web Texturizer</title>
<script language="JavaScript">
function switchurl(frm, type) {
    parent.content.location = "/cgi-bin/webtext-
dev/url_text.pl?url="+frm.T1.value+"&type="+type+"&keywords
="+frm.T2.value+"&type="+type+"&count="+frm.C2.value;}
function newurl(url) {
    document.frm1.T1.value = url;
    parent.content.location = "/cgi-bin/webtext-
dev/url_text.pl?url="+url+"&type=text";
}
</script></head>
<body bgcolor="#86A2FC" TEXT="WHITE">
<form name=frm1 ACTION="/cgi-bin/webtext-dev/url_text.pl"
METHOD="GET">
    <table width="100%" border="0" bgcolor="86A2FC"
height="34">
        <tr><td width="26%" height="29"><b><font face="Verdana,
Arial, Helvetica, sans-serif">Web</font>
            <font face="Verdana, Arial, Helvetica, sans-
serif">
Texturizer</font></b></td>
            <td width="19%" height="29"> <p align="right">Enter
URL:      </td>
            <td width="16%" height="29"><input type="text"
value="http://" name="T1" width=30 size="30"></td>
            <td width="8%" height="29"> <p align="right">Keywords:</td>
            <td width="11%" height="29"><input type="text" name="T2"
width=20 size="20"></td>
            <td width="5%" height="29"><p align="right">Count: <td>
            <td width="4%" height="29"><input type="text" value="10"
name="C2" width=10 size="5">      </td>
            <td width="11%" height="29"> <input type="button"
value="Display Text" name="B2"
onclick="switchurl(this.form, 'text'); return false">
        </td> </tr> </table></form>
<SCRIPT LANGUAGE="JavaScript">
```

```
document.frm1.T1.focus();
function keydownText() {
    if (event.keyCode == 13) {
        switchurl(document.frm1,'text');
        return false;
    }return true;}
document.frm1.T1.onkeydown = keydownText;
document.frm1.T2.onkeydown = keydownText;
</SCRIPT>
</body></html>
```

```

        // alert("setParNo("+n+")");
        parmax = n;
        parcurrent = 0;
        document.inp.parcurrent.value =
parent.content.kvalue(parcurrent);
    }
    // This is called when someone clicks on the paragraph on
    the frame
    function setParLink(n,m) {
        // alert("setParLink("+n+", "+m+")");
        parmaxlinks = m;
        linkcurrent = n;
        parlink = 0;
        document.inp.parlink.value = ""+n;
        // Calling this function will cause the value and
        paragraph to be set
        parent.content.setLocation(linkcurrent, parlink);}
    function setParValue(v, lnk) {
        // alert("setParValue("+v+")");
        document.inp.parvalue.value= v;
        document.inp.linkcurrent.value = lnk;
        parent.content.location.hash = lnk;
    }
</script></head>
<body bgcolor=#ffffff>
<div align="left">
<table border="0" cellpadding="0" cellspacing="0"
width="100%">
    <form name=inp>
        <input type=hidden size=1 name=parcurrent><br>
    <p>Paragraph Similarity Measure:
    Paragraph: <input type=text size=6 name=parlink><br>
    Paragraph viewed: <input type=text size=6
    name=linkcurrent><br>
    Measure: <input type=text size=14 width=14
    name=parvalue><br> <br>
    <table width="160">    <tr> <td><a href="#"
    onclick="nextlink(document.formname, -1);return false"></a></td><td><a href="#"
    onClick="nextlink(document.formname, 1);return false"></a></td>    </tr>    </table>
    <br>        To change the paragraph measured against,
    click on the paragraph number on the display to the right
    (e.g. 'P3').
</form> <p>&nbsp;  </td></tr> </table></div></body></html>

```

## CONTENT CODE

content.html

```
#Sample code of the content page when the user enters a url
#example: http://www.textseem.ehost4u.com/webtext-
dev/docs/related.html
Content-Type: text/html; charset=ISO-8859-1
<HTML><HEAD><STYLE>
BODY
{
    BACKGROUND-COLOR: white;
    FONT-FAMILY: Verdana, Arial, Helvetica;
    FONT-SIZE: 8pt;
    MARGIN: 4px
    VLINK: Blue;
    ALINK: Blue;
    LINK: grey;
}
TD
{
    FONT-FAMILY: Verdana, Arial, Helvetica;
    FONT-SIZE: 8pt
}
.Bg1
{
    BACKGROUND-COLOR: #ffffff;
}
.Bg2{
    # BACKGROUND-COLOR: #eeeecc;
    # FONT-SIZE: 10pt;
    BACKGROUND-COLOR: #dddddd;
    FONT-SIZE: 8pt;
}
.Bg3
{
    BACKGROUND-COLOR: #ffffff;
    FONT-SIZE: 8pt;
}
.Bg4
{
    BACKGROUND-COLOR: #CCCCCC;
    FONT-SIZE: 8pt;
}
.Bg5
{
```

```

    BACKGROUND-COLOR: #ffffff;
    FONT-SIZE: 8pt;
}
.Bg6
{
    BACKGROUND-COLOR: black;
    FONT-SIZE: 8pt;
}
a.thelink:link {
    color : #dddddd;
    text-decoration : none;
    font-size : 10px;
    font-family : verdana;
}
    a.thelink:active {
        color : #dddddd;
        font-size : 10px;
        text-decoration : none;
        font-family : verdana;
    }
    a.thelink:visited {
        color : #dddddd;
        font-size : 10px;
        text-decoration : none;
        font-weight : bold;
        font-family : verdana;
    }
}
a.thelink:hover {
    color : Blue;
    font-size : 10px;
    text-decoration : none;
    font-family : verdana;
}
</STYLE>
<script language=javascript>
var klist = new Array("par1","par2");
parent.menu.setParNo(klist.length);
function kvalue(n) {
    return klist[n];
}
var plist = new Array();
var vlist = new Array();
plist[1] = new Array("p1","p2");
vlist[1] = new Array(1.000000,0.312349);
plist[2] = new Array("p2","p1");

```

```

vlist[2] = new Array(1.000000,0.312349);
function pswitch(par) {
//  alert("pswitch -- "+par);
    parent.menu.setParLink(par, plist[par].length);
}
//var currentParagraph=null;
function setLocation(par, n) {
//  alert("setLocation -- "+par+" "+n);
//  alert("Hash set " + plist[par][n]);
    paranum= plist[par][n];
    parent.menu.setParValue(vlist[par][n], plist[par][n]);
    // ADDED BELOW
// var p=document.getElementsByName(plist[par][n]);
//alert(p);
// alert(p.innerHTML);// tell me what this says and if the
paragraph you want is in there
    //p.style.color="red";
// if (currentParagraph) {
//  currentParagraph.style.color="black";
// }
//currentParagraph=p;

//  alert("testing"+paranum)
    document.getElementById("tr"+paranum).setAttribute("cl
assName", "Bg5")
    document.getElementById("trtwo"+paranum).setAttribute(
"classname", "Bg4")
}

function linkto(url) {
    parent.nav.newurl(url);
}

</script>

</HEAD>

<BODY alink="#ffffff" link="#ffffff" vlink="#ffffff">

<TABLE>
</TABLE>
<TABLE cellpadding=0 cellspacing=0>
<TR ID=trp1 class=Bg1><TD><A NAME=p1 class=thelink
HREF="javascript:pswitch(1)" >. </TD></TR>

```

<TR ID=trtwop1 class=Bg1><TD><B>Contribute</B> as much as you can to employer-sponsored <B>retirement</B> plans. It's easy and convenient. You typically get a <B>tax</B> break on <B>contributions</B> so putting a dollar toward <B>retirement</B> means giving up perhaps only 60 or 70 cents in spending power now. If your employer <B>offers</B> to <B>match</B> <B>contributions</B>, put in at least enough to get the entire <B>match</B>. If you don't, you're passing up a free pay raise.</A></TD></TR>

<TR ID=trp2 class=Bg1><TD><A NAME=p2 class=thelink HREF="javascript:pswitch(2)" >. </TD></TR>

<TR ID=trtwop2 class=Bg1><TD>Everyone is eligible to <B>contribute</B> to an <B>IRA</B>. The only question is which type you should choose a <B>traditional</B> <B>IRA</B> or a <B>Roth</B> <B>IRA</B>. The answer <B>depends</B> on your financial situation. A <B>traditional</B> <B>IRA</B> may or may not <B>offer</B> an up-front <B>tax</B> <B>deduction</B> on your <B>contributions</B>, <B>depending</B> on your income, but it always <B>offers</B> <B>tax</B>-deferred investment growth. A <B>Roth</B> <B>IRA</B> is never <B>deductible</B>, but it <B>offers</B> <B>tax</B>-free withdrawals during your <B>retirement</B>.</A></TD></TR>

</TABLE>

<P><table cellpadding=1 cellspacing=1 border=0>  
<tr class=bg3 ><td><a href="#p1">p1</a></td><td><font color=> 1.00</font></td><td><font color="blue">  
0.31</font></td><tr class=bg3 ><td><a href="#p2">p2</a></td><td><font color="blue">  
0.31</font></td><td><font color=> 1.00</font></td></tr>  
</table>

<P><table cellpadding=1 cellspacing=1 border=0>  
<tr  
class=bg3><td></td><td>tax<td>plans<td>deduct<td>choose a<td>  
>employer-  
sponsored<td>question<td>free<td>offer<td>answer<td>roth<td>  
>raise<td>spending<td>eligible<td>convenient<td>means<td>de  
pend<td>power<td>break<td>dollar<td>match<td>giving<td>easy  
<td>contribut<td>withdrawals<td>income<td>typically<td>grow  
th<td>ira<td>don't<td>entire<td>tax-free<td>type<td>tax-  
deferred<td>pay<td>traditional<td>investment<td>passing<td>  
situation<td>cents<td>putting<td>retirement<td>employer<td>  
financial</td></tr>  
<tr  
class=bg3><td>p1</td><td>1<td>1<td>0<td>0<td>1<td>0<td>1<td>





## REFERENCES

1. Tom Christiansen & Nathan Torkington, *Perl Cookbook*., O' Reilly & Associates, Inc., 1998.
2. William W. Cohen, *Recognizing Structure in Web Pages using Similarity Queries*, American Association for Artificial Intelligence, 1999; pp 60-66.
3. William W. Cohen, *WHIRL: A word-based information representation language*, American Association for Artificial Intelligence, Vol. 118, 2000; pp 163-196.
4. David Flanagan, *JavaScript: The Definitive Guide*, O' Reilly & Associates, Inc., 1998.
5. Jeffery Friedl, *Mastering Regular Expressions*, O' Reilly & Associates, Inc., 1998.
6. M. F. Porter, *An algorithm for suffix stripping*, Program, Vol.14 No. 3, July 1980; pp 130-137.
7. Randal L. Schwartz and Tom Christiansen, *Learning Perl Second Edition*, O' Reilly & Associates, Inc., 1997.
8. Kerstin Voigt, *A Tool for Individualized Information Navigation That Adapts to User Domain Expertise*, American Association for Artificial Intelligence, 1997; pp 1-18.
9. Kerstin Voigt, *SKIPPER: A Tool that Lets Browsers Adapt to Changes in Document Relevance to Its User*, American Association for Artificial Intelligence, 1996; pp 1-17.
10. Kerstin Voigt, *Reasoning about Changes and Uncertainty in Browser Customization*, American Association for Artificial Intelligence, 1996 pp 1-6.
11. Kerstin Voigt, *Sacrificing vs. Salvaging Coherence: An Issue for Adaptive Agents in Information Navigation*, American Association for Artificial Intelligence, 1995; pp 1-12.

12. Clinton Wong, *Web Client Programming with Perl*, O'Reilly & Associates, Inc., 1998.