Direct multidisplay for web document repositories

by

Zhong Gu

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Major Professor: Daniel Berleant

Iowa State University

Ames, Iowa

2001

Graduate College
Iowa State University

This is to certify that the Master's thesis of

Zhong Gu

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

ABSTRACT

As the popularity of the Internet grows, the information on the Internet is increasing as well. Search engines are important tools to help people retrieve information of interest from the huge amount of documents. However, currently used search engines return long lists of URLs. Users have to click each URL to download the actual document and check its content and click the back button to access other URLs if the answer is not found in the current URL. This process is both labor intensive and time consuming. Multibrowser, a program that addresses this problem, is presented in this thesis. Multibrowser combines the advantages of multidisplay and direct display to present a more efficient user computer interface. First, the system downloads the actual documents according to the list of URLs returned by a standard search engine and saves the documents on the local disk. Second, the system converts the documents into n-gram vectors and clusters them into three groups according to the n-gram vectors. Then each document is assigned a color according to its position in relation to the cluster centroids. Also, each paragraph is linked with other paragraphs which have similar contents. Last, the documents are presented to the users using Multidisplay, where each document has a corresponding color bar. Users can look through the content of several documents at the same time and see the similarity among them by the colors bars; users can then retrieve the most similar paragraphs to a certain paragraph by clicking its "find similar" link. In addition, the investigation of hash tables for our text processing shows that the hash table size has to be chosen very carefully to avoid undesirably large collision probability. Some good values are suggested based on our experiments.

1. INTRODUCTION

Nowadays, the WWW has been becoming an inseparable component of our society and is a necessary part of our daily life. Actually, the WWW has a history of only about ten years. This chapter will give a brief review of the WWW and of search engines, and then discuss the need to improve the performance of search engines.

1.1 The Origin of the WWW

WWW is built on a foundation of HTTP (Hypertext Transfer Protocol) and HTML. HTTP was first introduced in 1989 by Tim Berners-Lee of CERN, the European Laboratory for Particle Physics. HTTP offers a stateless connection that allows the authors of Web pages to link pages containing multimedia files. It applies the client-server model: the server responds to the client when it receives a request from the client.

The first WWW browser was Mosaic, which was developed at NCSA (National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign). Mosaic is a client software system which communicates with WWW servers and translates Web page HTML codes and text into a display for the user. Browsers provide an easy way to view and navigate the Web. When users click on a hypertext link, the browser can retrieve the appropriate URL by communicating with the appropriate Web server.

1.2 History of Automated Web Searching

When the Web first became available to the public, the only way to navigate it was by "surfing." That is, a user began with a known page and launched forth from the links it contained, browsing until something of interest had been found. In the early days, there were only several hundred HTTP servers on the Internet, so this method could work. Due to the

rapid growth of the Web both in the numbers and kinds of resources available, it became necessary to find a better way of organizing information on the Web.

In October 1993, ALIWEB (Archie-Like Indexing of the WEB) was developed by a team led by Martijn Koster. ALIWEB combines human and automated effort to generate the index. In ALIWEB, a web-master has to create and maintain a special ALIWEB file which describes the contents of the site on his/her server. ALIWEB would then use fully automated robot software to travel the Web searching for these special files. The robot software will compile the names of these files into an index. Then the users can use these indexes to ease information hunting.

The code for ALIWEB's automated robots was placed on the Internet after the creation of ALIWEB. Many more complicated robots were created by revising these codes, which include spiders, worms, crawlers and intelligent agents. These new generation robots retrieve actual Web pages for indexing instead of specially designed files.

WWWW (World Wide Web Worm), or W4 was one of the first publicly accessible search tools to appear on the Web. The Worm used its robots to discover documents and index their URLs (Uniform Resource Locators) and <TITLE>s. The Worm was based on a theory similar to citation searching. It treated links as citations to Web pages. The Worm's database contained only Web pages that were linked from other Web pages, in an effort to capitalize on an author's judgment that another page was worth referencing. It indexed the URLs, and disassembled them into component words that were stored together so that users could retrieve the associated pages using keyword searches. It was hoped that, by limiting the index exclusively to URLs and <TITLE>s, the Worm would be able to keep up with the rapid growth of the Web without using a prohibitively large amount of resources.

The Worm was a big improvement over its predecessors in Web Retrieval technology. It offered a way to create a database of Web pages without external human intervention. A user search would produce results in the form of a unique set of <TITLE>s and URLs hot-linked to the original documents on their home servers. The requirement for the Worm is that Web page authors should put sufficiently descriptive information in a document's <TITLE> or URL.

Then Web Crawler and Lycos were developed, which established the pattern for most of the currently used search engines. More new search engines were developed after that and they are available to serve users.

Though many different search engines have different implementations and can return different search results, the basic working mechanisms are the same.

1.3 How Search Engines Work

All of the modern search engines have following basic functions:

- Finding Web pages
- Harvesting Web pages and building an index
- Searching the index with a user query
- Providing a user interface

1.3.1 Finding Web Pages

A Web robot is defined in the WWW Robot FAQ as ... " a program that automatically traverses the Web's hypertext structure by retrieving a document and recursively retrieving all documents that are referenced." It is used to collect web pages.

All robots begin searching from some active page on the Web, preferably one with a variety of links to follow. Beginning with the first link on this active page, a robot can reach other pages by employing either a depth-first or a breadth-first strategy.

A depth-first strategy would send a robot to the page attached to the first link on the initial page, then to the page on the first link of the second page, then to the first link on the third page, and so on. The robot evaluates each link based on some criteria, such as whether the page contains indexable text or whether or not the robot has already encountered the page before, and then it can choose either to store the URL or to discard it. When the robot reaches a dead end, either by attempting to follow a link to a page that is no longer available or by accessing a page that contains no outward links, it takes one step back to the page it last visited and follows the next available link.

In the breadth-first strategy, the robot takes one step backward after each step forward. It follows the first link on the initial page, then retreats to follow the second link on that same page, as shown in Figure 1-1. It continues in this manner until it has gathered or discarded all of the URLs of the links on the first page, then it moves on to the page associated with the first link.

After it builds a database of URLs to work with, it can choose a link from the URL database to begin the next excursion.

1.3.2 Harvesting Web Pages and Building an Index

When all the available URLs have been saved in a database, another subsystem (harvester robot) will be used to build and search the index. The harvester robot revisits each Web page the first robot discovered, extracts part or all of the text from the page, breaks it

into component words, and integrates it into a master index of words. Most search tools also create a separate database containing records similar to catalog cards. Each record is composed of a Web page URL, title, and summary. When a user search retrieves a URL, the search tool presents this summary record to the user. These master indexes are updated on a daily or weekly basis.

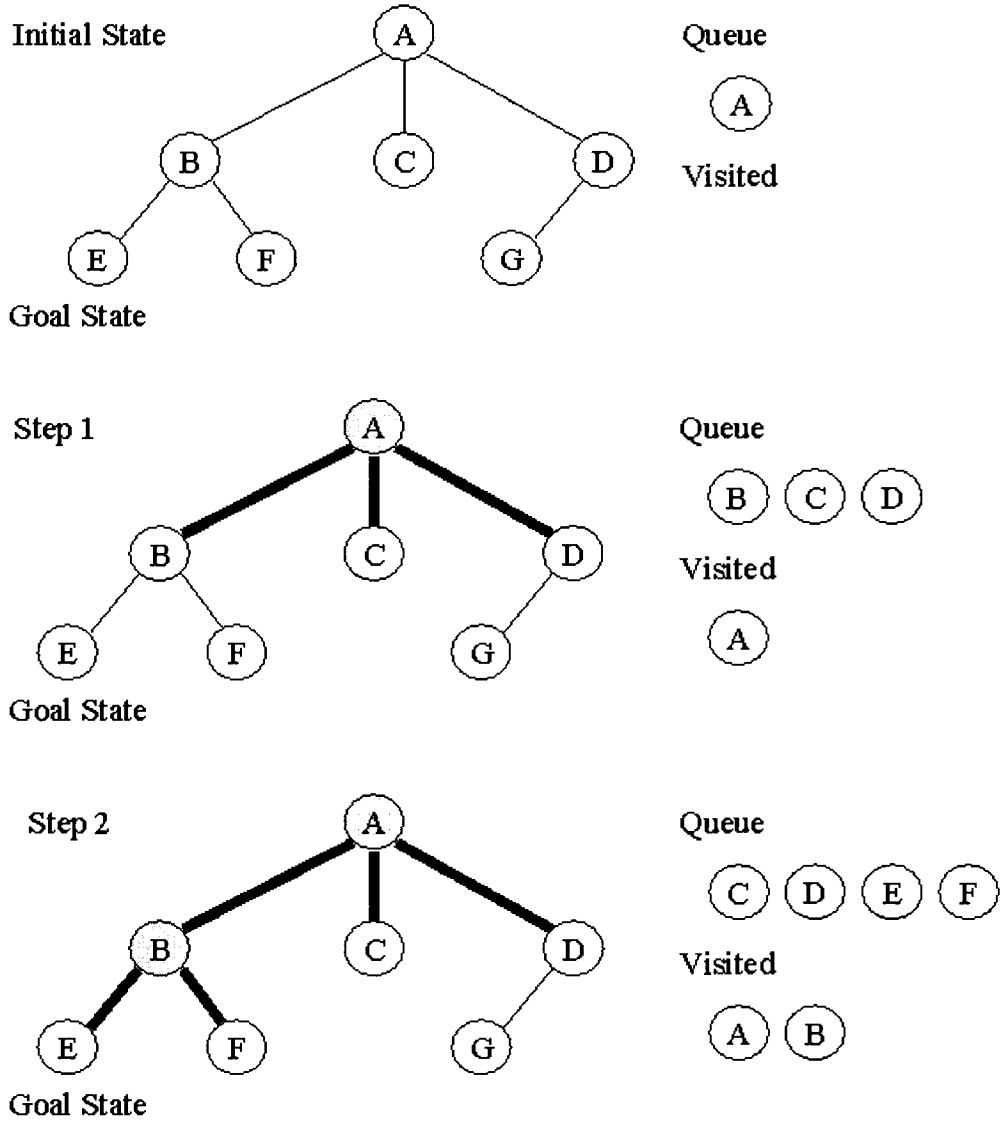


Figure 1-1. Breadth-first Search Example

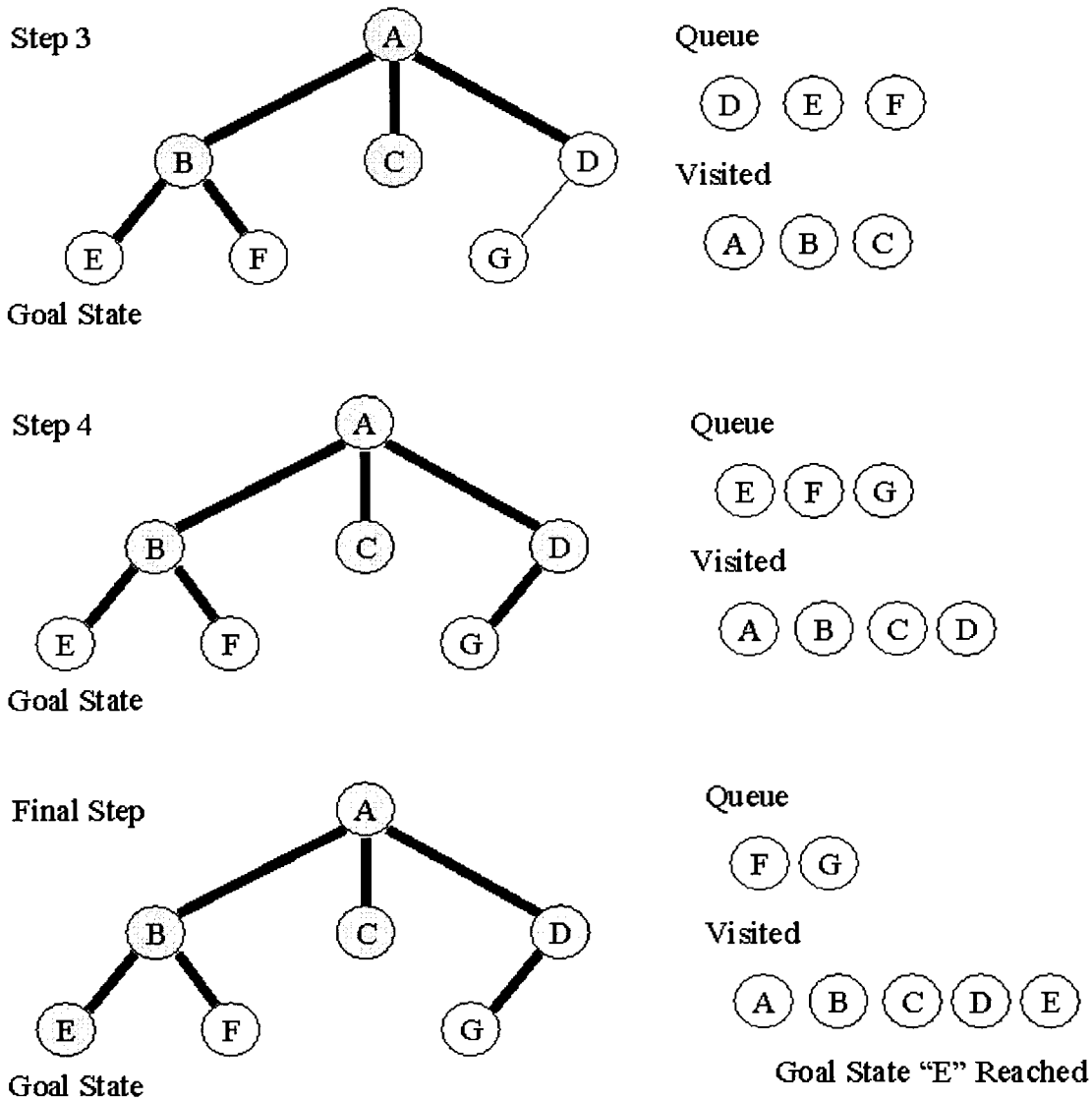


Figure 1-1. Continued

Words are the basic units of the index. Each word entry in the index has four fields:

- The URL of a page containing the word
- The word's position in the page
- The total number of words in the URL or <TITLE>

- The field in which the word occurs

Most web search tools index the URL and <TITLE> parts, and at least part of the text body from a web page. As the harvester robot reads each word on the Web page, it creates an entry in the database. Web search tools also use other criteria to determine what constitutes a word. Some robots do not index words less than three characters long, and automatically omit some common stop-words, such as "to" and "in." Some other statistics for each word can be added too.

Once a query is input, a search engine breaks the user query into its component pieces. The search engine checks the index for each word of the query individually. As it receives the results for each word, it creates a temporary table containing the entries from the index database for this word to help assemble and order the final list of results. Then the search engine will merge these individual tables into another table according to a ranking algorithm. A ranking algorithm determines the order of the URLs which will be returned to the user. Then the search engine returns the URLs together with their summaries to the user.

1.3.3 User Interface

The user interface is the part of search engine used to directly interact with users and is the part of the search engine that the user sees. It dictates what the user may type and what results appear on screen. The user interface can be divide into two parts: one is the interface used to input the search key words and the other one is the output interface for displaying the search results.

1.3.3.1 Interface for Entering Search Key Words

Search engines get the key words input by users, then retrieve the appropriate information according to the key words. The simplest is just a text field where words can be

input. Some advanced ones support Boolean expressions on key words.

Figure 1-2 is an example from northernlight.com. Queries are inputted into a textfield. When the user clicks the search button, the queries will be submitted to the search engine and the search process begins.

1.3.3.2 Interface for Displaying the Search Results

When a search engine finds the information, it ranks the items according to some criteria and presents the result to the users. The user's custom results page is typically a set of citations and each is composed of a hypertext link in the form of a Title or URL together with some other information. The results appear in ranked order. Users scroll down the page to view hits. Users have to click URLs to download the documents.

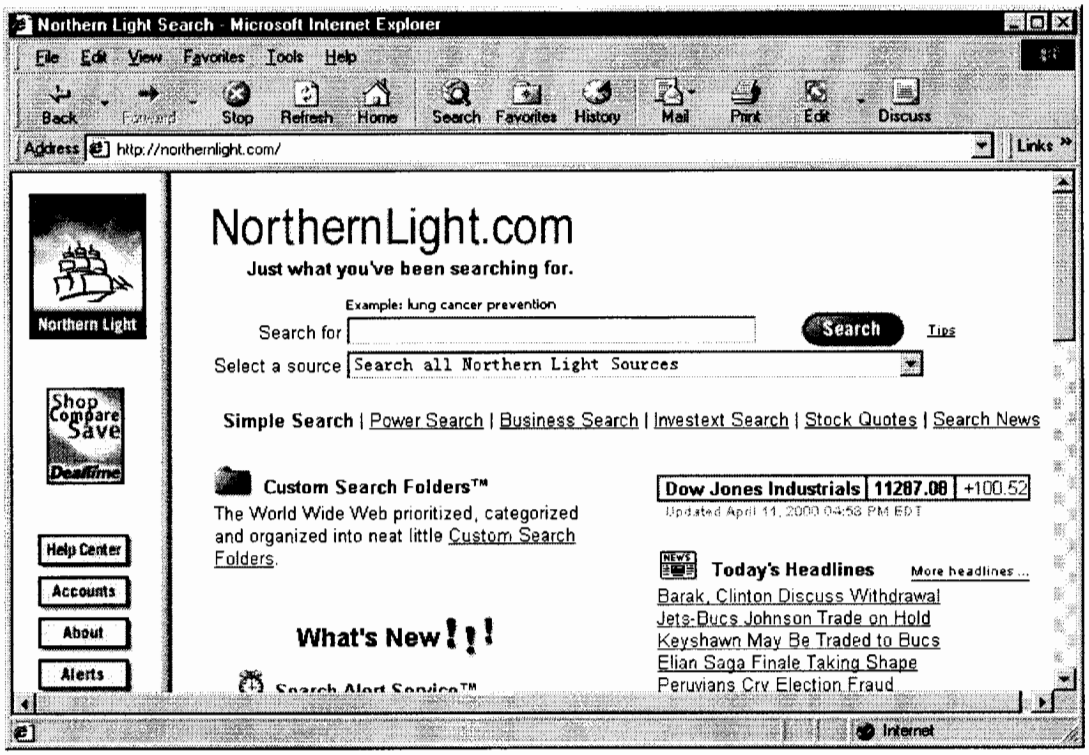


Figure 1-2. query input window

If a page linked to the esults has been moved or deleted, or if its home server is unavailable or too busy, or if any of a number of other mishaps has occurred, the user may not be able to reach the page.

Figure 1-3 is an example of search results from northernlight.com.

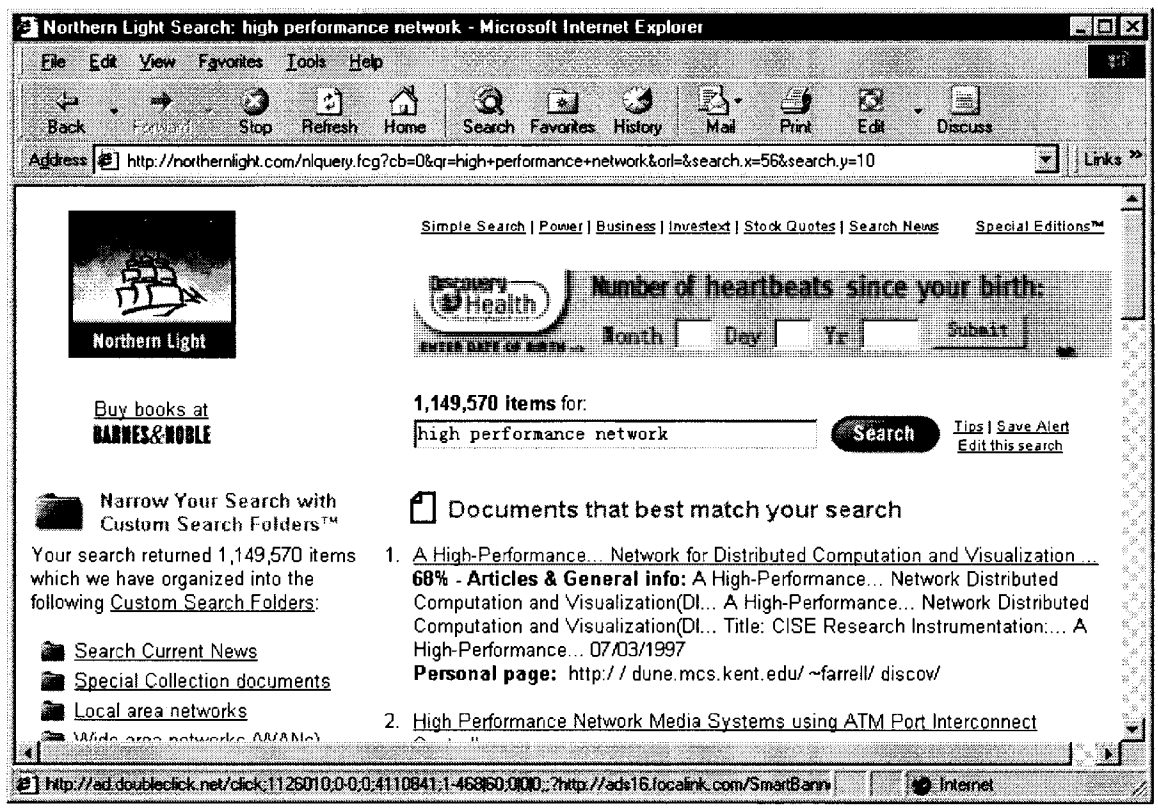


Figure 1-3. An example display of Search result

1.4 Drawbacks of Current Search Engines

Current search engines have serious limitations. One of the major limitations is that the returned results can contain many links pointing to irrelevant documents. One way to reduce the number of irrelevant links is to provide a predetermined taxonomy, and search only among documents pre-categorized as being in the desired taxonomic category. Many search

engines have applied this way to organize the links. This is only a partial solution as it is presently labor intensive to create the taxonomy and locks users into a one-size-fits-all taxonomic structure that might not be suitable for a given information seeking need. The standard way to solve the other major problem, excessive length of returned URL lists, is to provide the items in the returned lists in the order of their estimated relevance. While relevance ordering is useful, it is well known that the quality of the resulting orderings is imperfect. Even many ordering can have erratic performance on specific searches.

1.5 Improving Search Engines

In recent years, many new strategies are presented to optimize the results of the search engines. The retrieved documents are clustered and presented to the users using some visualization techniques. It facilitates the users in finding the relatedness among documents because the documents are clustered and visualized by the relevance of their contents. For example, in Mukherjea's system (1999), a Bulls-Eye visualization is used. In this system, the document of interest (main document) is located at the center and other documents are around it. The positions of the documents are determined by the similarity between the main document and another document. Grouper (Zamir 1999) retrieves the documents and clusters them using Suffix Tree Clustering (STC), and displays the results in different groups.

Our system, Multibrowser, applies multidisplay and direct display to enhance the efficiency of the user interface. After the documents are downloaded to the local disk, they are clustered into three groups. Each document is assigned a color according to its position in the clusters. Furthermore, each paragraph has a link to several other paragraphs which are most similar to it. The display window shows several documents simultaneously and users can find the similarity among different documents by the color bars associated with the

documents. Users can also click the link associated with a particular paragraph to retrieve several paragraphs which are most similar to that paragraph.

1.6 Thesis Organization

The remainder of this thesis is organized as follows: first, the concepts of multidisplay and direct display will be presented in Chapter 2; then the details of the multibrowser system are discussed in Chapter 3. Some discussion about hash tables for n-gram based processing is discussed in Chapter 4; and a conclusion is given in Chapter 5. An appendix discusses our research work on using software engineering concepts to enhance learning.

2 DIRECT DISPLAY AND MULTIDISPLAY

The user interface is one of the most important components of modern computer applications. Search engines, like many other applications, have to present the desired information to users in a friendly way.

Nowadays, the Web is becoming more and more popular. As a result, the information on the web is increasing exponentially. The information returned by a search engine is huge and it is often not easy to find the desired information from such huge amount of documents. So, how to provide users a more user-friendly interface is a key issue to improve the efficiency of a search engine.

Most of the current search engines return a long list of URLs. Users have to click URLs to retrieve the corresponding web pages and then find the useful information. This procedure involves lots of cognitive overhead and is labor intensive because of the following reasons. First, users have to choose which URLs to check by reading meta-information returned by a search engine, which are usually very short and ambiguous. Second, users have to access a web page by clicking a URL and if there is not enough information on this page, the users have to click the "Back" button to return to the search engine result pages before going to the next URL.

Visualizing the search results is one way to give users a clearer idea about the search results. In recent years, diverse visualization techniques have been studied to help navigate through large numbers of documents.

Multibrowser is one way to provide a more user friendly interface, which combines the advantages of direct display and multidisplay, so it is more efficient than the user interface of

traditional search engines. In this chapter, the ideas of direct display and multidisplay are introduced first, and then multibrowser will be presented.

2.1 Direct Display Vs. Indirect Display

Direct display is contrasted with indirect display. Their definitions are as follows:

Definition: direct display is the presentation of actual content (Berleant 2001).

In direct display, a user can see the content of the documents.

Definition: Indirect display is the presentation of meta-content (Berleant 2001).

In indirect display, a user cannot see the real content of the documents. What users see is meta-information about the documents, such as the authors of the documents, sizes of the documents, last update times, a short passage, and so on. All major search engines use indirect display to present the search result to the users. Users have to decide which documents should be downloaded by checking the meta-information.

2.1.1 Comparison Between Direct Display and Indirect Display

Because what indirect display needs is just the meta-information of documents, it needs less network resources than direct display. But it is difficult for users to make judgments from the meta-information about which documents are preferable, because the meta-information cannot always represent the main ideas of the documents. In contrast, direct display presents users the contents of the documents, so users can make the decisions in the first place. Table 2-1 shows some of the pros and cons of direct display and indirect display.

Nowadays, the available network resources, such as bandwidth and storage capability, etc, are increasing rapidly; so direct display can be employed in more and more applications.

Table 2-1. Relative advantages and disadvantages of direct and indirect display strategies. Characteristics upon which a comparison is made are listed in the left column.

Characteristic	**Direct display**	**Indirect display**
Ease of access to content	Yes, content is displayed	No, content is not immediately displayed
Clarity of display	Higher since actual content is displayed	No, lower since abstractions lose information
Interface actions required	Fewer since content is immediately present	No, higher since content must be explicitly requested
Cognitive overhead	Less since user can absorb information without obstacles	No, more since user must judge relevance of content based on meta-information, and navigate more
Utilization of future technological improvements to bandwidth and display real estate	Greater since more information is transmitted and displayed	Less since abstractions economize on transmission and display
Suitability for display of a limited number of items	Greater since if things can be shown directly it is often useful to do so	Less due to less need for abstraction when a limited number of information items are present
Summarization/preview function	Content not summarized	Yes, meta-content semantically compresses content
Bandwidth required for display	Higher since information is not abstracted	Lower since abstractions are usually smaller
Display footprint	More since full information is displayed	Less since less is displayed
Suitability for multidisplay of many items	Less since too many items won't fit in display area	Greater since abstractions reduce footprints of each item, allowing more to be displayed at once

2.2 Monodisplay Vs. Multidisplay

Direct display and indirect display concern what should be presented to users (content of documents or meta-information about documents); monodisplay and multidisplay handle the question of how to present multiple documents to users.

Definition: a monodisplay is the presentation of one item (Berleant 2001).

Definition: a multidisplay is a tiled, space-filling collage (Berleant 2001).

Monodisplay presents one document to users one time. Most search engines use monodisplay: users have to click one of the URL from the URL list returned by a search engine to retrieve the content of that document. If users want to read the content of another one, they have to click the back button to return the previous page to click another link.

On the other hand, a multidisplay presents users more than one documents simultaneously.

2.2.1 Comparison Between Monodisplay and Multidisplay

Monodisplay is easier to implement and it is enough if users just want to process one document. Multidisplay is more preferable if users want to access several documents. When a monodisplay system is used to display several documents, users have to perform some actions, such as clicking mouse buttons, to access every document. It is labor consuming and also interrupts thought flow. A multidisplay system can display more than one document in response to one action from a user, so users need fewer actions than in a monodisplay system.

Another important feature of multidisplay is that it can display the relationship among different documents. Users can find the wanted information faster because when a document

is found useful, other similar documents can be found easily with the help of the relationship indication.

Because multidisplay is superior to monodisplay, and direct display is better than indirect display, we combine multidisplay and direct display to generate a new system to browse multiple documents. This is the direct multidisplay approach.

2.3 Combination of Direct/Indirect Display and Mono/Multidisplay

There are four possible combinations of multi/monodisplay and direct/indirect display. They are indirect monodisplay, direct monodisplay, indirect multidisplay and direct multidisplay.

Indirect monodisplay is used to display information about one document.

Direct monodisplay is used to display the content of one document.

Indirect multidisplay is used to display several items at the same time with each item containing some information describing one document. Most of the current search engines apply the indirect multidisplay strategy to present users the search results.

Direct multidisplay displays the content of several documents simultaneously.

2.4 Direct Multidisplay and Multibrowser

First, let's review some of the disadvantages of the strategies existing in current search engines. Most of the current search engines return a long list of URLs to the users and there is limited descriptive information about the documents that are linked by the URLs. In order to find the desired information, users have to read the descriptive information, click a link to access a document, and go through the document to get an idea of this document. If users don't find the desired information, they have to return to the URL list page by clicking the "Back" button, then access another document, and so on. This process involves lots of

cognitive overhead because users have to make judgments to choose some relevant URLs to click on to access the real documents according to the meta-information. Also, they have to remember which ones have interesting stuff. This process is labor intensive and time consuming because users have to check so many URLs to find the desired information.

In view of above discussion, it's obvious that direct multidisplay is very helpful in overcoming the drawbacks of current search engines. Direct multidisplay can show the content of several documents at the same time, so users need fewer button clicks to check several documents. So, our multibrowser applies the direct multidisplay strategy.

The aim here is to make it easy to extract the information from documents. But there are some issues to be considered.

- Which documents should be shown simultaneously and how to navigate those documents. Our solution is to assign each document a color. The similarity of colors shows the similarity of the contents of documents, so users can easily find the relatedness among the documents according to the their colors. When a document is found interesting, users can easily find other ones with the similar content by checking the documents with the similar color. See figure 3-2.
- How to navigate in the documents. Before talking about the solution for this one, paragraph processing is discussed first.

The aim of paragraph processing is to find the degree of relationship among paragraphs within the same document or other documents. It is very common that some paragraphs with similar contents are in some documents whose contents are totally different. Sometimes, what users want is just a piece of information within one paragraph. It is very useful to find similar paragraphs in a set of documents, so users can easily find the paragraphs related to a

paragraph of interest. Multibrowser divides the documents into paragraphs. For each paragraph, the several most similar paragraphs are found and a link is attached to this paragraph. When the link is clicked, the multibrowser will present those most similar paragraphs together with the whole documents. See figure 3-2. This solves the navigation problem at the same time.

In next chapter, details of the multibrowser system will be introduced.

3 MULTIBROWSER

In previous chapters, the concepts of multidisplay and direct display have been introduced. Our system, multibrowser, is discussed thoroughly in this chapter.

3.1 Related Work

Document clustering has recently been investigated as a technique to deal with organizing documents retrieved by search engines on the Web. The intent of document clustering algorithms in this context is to try to group documents together based on their similarities. Given such clusters, users could find interesting documents more easily than otherwise, as well as understand the structure of the body of retrieved documents better.

Document clustering has traditionally been performed in batch mode, but recently the need to cluster results returned by search engines in real time has led to work on real time clustering. Such clustering methods process documents relatively superficially since deeper analyses cannot be done in real time. For example, Excite (www.excite.com) identifies potentially useful search terms for refining the search, based on their ability to define a cluster among the returned documents, and hence to weed out other documents. The NorthernLight search engine (www.northernlight.com) provides "Custom Folders" and the retrieved documents are organized in these folders. These folders are clusters based on several parameters of the documents, such as type, source, language or subject.

Grouper (Zamir and Etzioni 1999) works as a meta-search engine interface. It queries different search engines, and only looks at the top hits from each. Thus, its clusters are not affected by many low relevance documents, as is the case in NorthernLight's folders. It also can run on a client machine, making it a friendly component of distributed IR systems. It is also argued to be scalable. Grouper also relies on the concept of coherent clusters, meaning

that the clustering algorithm groups similar documents together without excluding a document from some reasonable cluster just because it is already in another cluster. Thus Grouper can generate overlapping clusters. Grouper uses Suffix Tree Clustering (STC) – a fast, incremental, linear time clustering algorithm that produces coherent clusters. Better known in the bioinformatics field, suffix trees have a variety of computational advantages, and are argued to provide results in Grouper that are both faster and more precise than previous algorithms.

Visualization techniques can assist users in dealing with document clusters efficiently. One class of visualizations groups documents sharing such attributes as size, publication date, forum, etc. Another class compares documents based on their content of keywords, n-grams, etc.

2CE developed a 3-D browser. This browser presents several documents simultaneously by displaying them on the different faces of a cube. The sides of the cube rotate when a user clicks on one face of the 3D cube, and the page being clicked becomes the primary page in the center of the screen.

Our present work uses a simple visualization abstraction of a document – a color – to allow convenient visual perception of the relatedness among documents which have passages presented and visible on the screen. N-grams are used in our present work for document comparison and hence for clustering. Damashek (1995) describes using n-grams for document comparison. Damashek explicitly ignores collisions. Our work does also, but in addition, we investigate the issue of collisions. Cohen (1997) focuses on recursive hashing of n-grams as a way to improve computation speed, while the present work focuses on hashing issues relevant to memory usage.

3.2 System Overview

Basically, multibrowser is a direct multidisplay system which displays the contents of several documents at the same time. Documents are processed before being displayed to make it easy to find the relatedness among documents. The processing includes two steps: document-scale processing and paragraph-scale processing. In document-scale processing, documents are clustered into three groups and each document is assigned a color according to its position in these three groups. In paragraph processing, the N most similar paragraphs are found for each paragraph and a link is attached to this paragraph which retrieves the N most similar paragraphs when clicked. Here N is the number of subwindows on screen. The interface of the system displays several documents at the same time and users can see the relatedness among documents by their color. In addition users can easily retrieve the N most similar paragraphs to a certain paragraph by clicking the link attached to this paragraph. Another issue is that how to handle duplicated pages. Sometimes, several pages have the same or almost the same content, which is a kind of information redundancy. That means we don't want two paragraphs with the same content to appear on screen at the same time. Multibrowser accomplishes it by ensuring that any two of the N most similar paragraphs have a distance greater than a threshold. The system can find different sets of paragraphs depending on the threshold.

It works as follows:

1) Some key words are inputted.

2) The system contacts a standard search engine.

3) It retrieves the corresponding URL list returned by the standard search engine.

4) It downloads the documents from their URLs.

5) It converts each document to a vector according its n-gram values and this n-gram vector is used in the following step to gauge the similarity of documents and paragraphs.

6) It clusters documents and assigns a color to each document to represent its position in the clusters and generates the corresponding color files.

7) It orders the paragraphs of documents according to their n-gram values and finds the N most similar paragraphs to each paragraph.

8) It generates new HTML pages.

The details of system are discussed soon. First, the idea of n-grams is introduced because it is the basic tool used to gauge the similarity of the documents.

3.3 N-grams

3.3.1 What Is an N-Gram?

An n-gram is n consecutive characters. A document can be represented by the distribution of the n-grams in it. The distributions of n-grams of different documents can be used to compare the similarity among the documents.

In more detail, a document can be represented as a vector whose components are the numbers of occurrences of the distinct n-grams. But the problem is that the dimension of the corresponding vector is large. For example, when 5-grams are used and all the 256 one-byte characters are considered, the number of possible 5-grams is $256^5 = 2^{40}$ which occupies too much memory. So we have to figure out ways to overcome it. Fortunately, most of the possible 5-grams almost never occur in real documents. So we can use a hash table to represent a document vector and this way saves lots of computer resources. If a reasonable hash table is used, the number of collisions introduced by hashing the n-grams that occur is small enough.

In our system, a document is converted to the vector representation by the following steps:

1) Let H be the size of the hash table and W an n-character wide window.

2) Construct an H-dimensional array A and initialize all the components to 0.

3) Move the window W one character along the document.

4) Convert the corresponding n-gram into a number.

For example, an n-gram gram can be converted to a number V by:

$$V = gram[0] + gram[1] * 256 + \cdots + gram[n-2] * 256^{n-2} + gram[n-1] * 256^{n-1}.$$

5) Use a hash function to get the value of this value.

6) Increase the corresponding component of the array A by 1.

7) If the end of the file is not reached, go back to step 3).

8) Return the n-gram vector.

3.3.2 Using N-Grams to Gauge the Similarity of Documents

After the documents have been converted to vectors, these vectors can be used to gauge the similarity among documents. These vectors support a cosine-based document similarity metric (Damashek 1995).

In Multibrower, the following metric is used to gauge the similarity between two documents. Suppose two documents have n-gram vectors V_1 and V_2 respectively and D is the hash table size. First these two vectors are normalized, that is,

$$W_i[j] = \frac{V_i[j]}{\sqrt{\sum_{k=1}^{D} V_i^2[k]}} \qquad 0 < j \le D$$

Where W_i is the normalized vector and $\sum_{k=1}^{D} W_i^2[k] = 1$.

Now, all documents (represented by the heads of vectors of n-gram hash codes) fall on the hypersurface of a hypersphere in the many-dimensional space whose axes are the hash codes in our space of possible hash codes.

Then a value S is calculated using the following equation:

$$S = \sum_{i=1}^{D} W_1[i] * W_2[i]$$

The more similar the documents, the larger the S. This is known as the cosine measure. S can be used to gauge the similarity of documents; higher values of S indicate greater similarity.

3.4 How the System Works

3.4.1 Step 1: Downloading the Documents

The system starts by accessing a standard Web search engine, giving it a query, and downloading the documents in the list of URLs returned by the search engine. This downloading, and subsequent processing, takes time. Users could let it run while doing other things or overnight, or librarians could use it to create browsable repositories which, once created, would be available for browsing interactively in real time.

3.4.2 Step 2: Clustering the Documents

Once downloaded, the documents are clustered. Later, the paragraphs will be subjected to an independent processing step which will directly support browsing from one paragraph to related paragraphs.

To cluster the documents, every document is converted into a corresponding vector of n-grams. These vectors are the input to the clustering algorithm.

In order to indicate the relatedness among documents visually, the documents are clustered into three groups which correspond to three dimensions named red, green and blue respectively. A document's coordinates along each dimension determine the contribution of the corresponding color to the associated colored bar which is assigned to every document. It means the similar documents will have similarly colored color bars.

Here the K-means Clustering Algorithm is used to cluster the document vectors.

Let $\{v_1, v_2, \ldots, v_N\}$ be the set of N vectors.

Let $c_1^{(j)}$, $c_2^{(j)}$, $c_3^{(j)}$ be the centers of three clusters in the jth iteration.

Let N_i be the number of vectors in cluster i, i=1, 2, 3.

Then

$$F_i^j = \sum_{k=1}^{N_i} \left\| v_k - c_i^j \right\|$$

represents the sum of squared distances of all vectors in a cluster to the cluster centroid.

The vectors are processed by following procedure:

Randomly choose 3 vectors from all of the vectors as the initial cluster centers $c_1^{(1)}$, $c_2^{(1)}$, $c_3^{(1)}$.

In the jth iteration:

Assign vectors $\{v_i\}_{i=1}^N$ to the 3 clusters using the minimum distance classification rule:

$v \in$ cluster m m=1, 2, 3.

if $\left\| v - c_m^{(j)} \right\| < \left\| v - c_p^{(j)} \right\|$

$\forall p \neq m$, $p, m = 1, 2, 3$.

Calculate the new cluster centers $c_i^{(j)}$

$$c_i^{(j+1)} = \frac{1}{N_i} \sum_{k=1}^{N_i} v_k \qquad \text{i} = 1, 2, 3.$$

(3) Stop if $F_i^{(j+1)} = F_i^{(j)}$, i = 1, 2, 3.

That means that the sum of the squared distances of all vectors in a cluster to the new cluster centroid $c_i^{(j+1)}$ is minimized.

Now all the documents are clustered into three groups and we can get the three centroids of the clusters. Then a document's distances to the three centroids will determine the contribution of the corresponding color to the colored horizontal bar which is displayed in conjunction with the document. See figure 3-1.

3.4.3 Step 3: Generating Color Codes for the Documents

For each document, an RGB color is generated based on the cosine of the corresponding angles to the centroids of clusters R, G, and B. This color is the background color of a small HTML file to be used as a colored tag for the corresponding document. See figure 3-1. When a document is eventually displayed, it will be displayed starting at the beginning of some paragraph in it, and the color tag of the document will be displayed in a horizontal bar above the document. Thus documents that are measured to be similar will be displayed with similarly colored horizontal bars allowing readers to quickly and visually determine relatedness among documents displayed simultaneously in different frames (Figure 3-2). Each document's horizontal bar also contains the URL from which the document was downloaded, increasing the information density of the color tag.

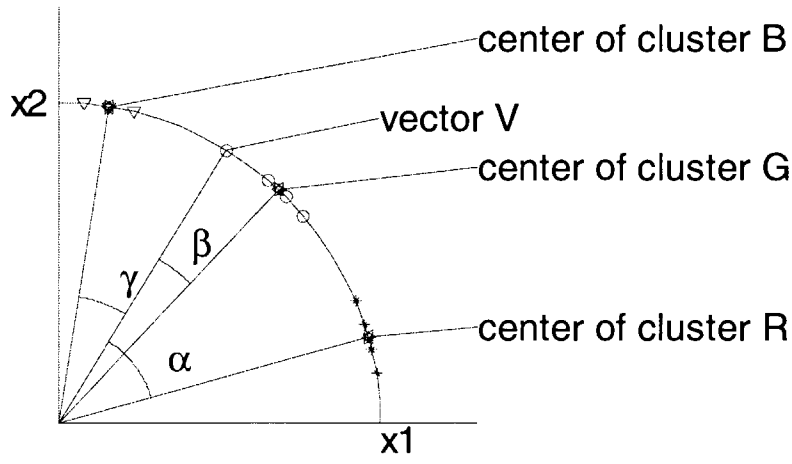


Figure 3-1. An example of document clusters, simplified to 2 dimensions. The cluster centroids are named R, G, and B. A document vector V's degree of relatedness to each centroid is determined by the angles by α, β and γ between it and the respective centroids and is proportional to the cosine of those angles. Only a quadrant is needed because all vector components are positive.

3.4.4 Step 4: Paragraph-Scale Processing

In this step, each paragraph in the set of documents is represented by an n-gram vector as described in the above paragraphs. And for each paragraph, the M most similar paragraphs are found, where M is the number of subwindows used by the system. Thus, users will be able to easily find the most similar paragraphs to a certain paragraph. The most similar paragraphs can be found by standard sorting algorithms from the set of paragraph vectors.

3.4.5 Step 5: Reformatting the Documents

In order to facilitate use, some additional information has to be added to the original documents. For example, each paragraph in each document can be provided with an icon which, when clicked, retrieves the six most closely related paragraphs in the set of documents and sends to the user's browser their containing documents, specified to display at the beginnings of the retrieved paragraphs.

Currently, a set of HTML frames is generated and each paragraph's containing document is shown in a frame. The original documents are revised to add a hyperlink to each paragraph. When such a link is clicked, the corresponding html frame files will be retrieved. To find the paragraphs most related to any given paragraph, the same n-gram based vectorization process followed for whole documents will be used. Figure 3-2 shows an example display.

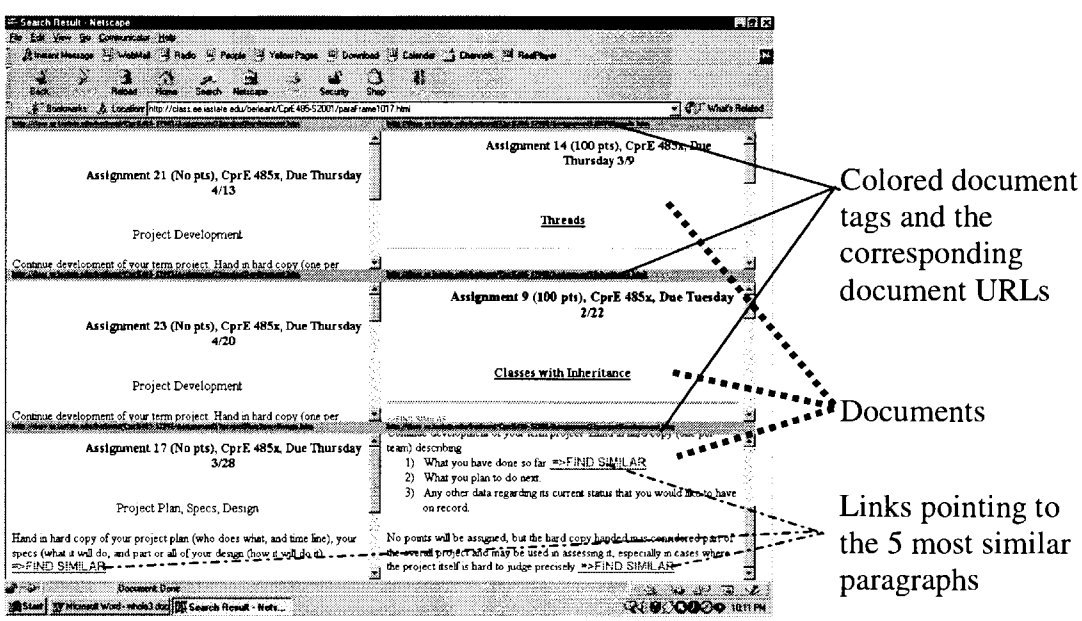


Figure 3-2. Example display

3.5 Some Problems Involved in the System

The first problem occurred when the documents were converted to n-gram vectors. The HTML tags in the documents have to be removed before they are converted to n-gram vectors. Consecutive carriage returns are removed to decrease the effect of the formatting information.

Another problem occurred when the downloaded pages are displayed. Some homepages have a <base href="http://someServer"> tag. A browser tries to find all the links on the current page using the http://someServer as a base, when it finds such a tag on the current page. Also, some pages have missing script source and Netscape cannot handle this. Our solution is removing such tags.

3.6 Future Work

The system has to download the documents and preprocess them and both of the procedures take time. Therefore it cannot be used like a usual search engine because of the long delay.

The solution to overcoming the long delay is using a repository, that is, the system downloads the documents, processes them and saves the information on local disk. So one of the most important issues is how to overcome the long delay of the multibrowser.

The system software is implemented with Java to increase its extensibility because Java is an object-oriented programming language. So it's very easy to extend the system to accommodate new features in future work. For example:

- Currently, the software only retrieves the search results from Altavista, but it can be extended to retrieve search results from other search engines by adding corresponding classes.

- Currently, the six most similar paragraphs are found for each paragraph. This number can be changed to other values. Finding the N most similar paragraphs is implemented in a sorting class. N is a parameter of this class, which can be specified to change the number of most similar paragraphs found.
- Currently, 5-grams are used to gauge the similarity of the documents and paragraphs. It can be changed to some other value of n by changing a parameter in the system.

In the next chapter, we investigate some issues that arise in using n-grams as we have used them here.

4 HASH TABLE SIZES FOR STORING N-GRAMS FOR TEXT PROCESSING

4.1 Introduction

Natural language text processing applications such as information retrieval, document comparison, and document clustering make extensive use of string comparisons. Because texts generally are constructed from words, such comparisons have frequently relied on comparing the word profiles of the texts. However pure word comparison has significant shortcomings. Words have different lengths leading to computational efficiency deficits in speed and memory relative to computations on constant-length strings. Furthermore, assessing similarity between different but related words (such as different inflections of the same word root) can be important in text processing, and the naïve approach of recording a match between two words only if they are fully identical frequently supports merely a baseline performance level from which significant improvement can be sought.

Stemming algorithms can alleviate the word match problem, but at the cost of additional computation. N-grams can both alleviate the word match problem and support improved computational efficiency compared to word based processing, because n-grams are simply strings of length n. Thus 4-grams all have length 4, 5-grams have length 5, etc. N-grams support partial matching when texts contain different but similar words because the similarities between the words cause the passages to have n-grams in common. For example, "computer" and "computation" are different words but share two 5-grams, "compu" and "omput".

4.1.1 Uses of N-grams

N-grams were investigated for tasks related to information Retrieval at least as early as 1979 (Suen). Since then they have been investigated in such tasks as language identification (Damashek 1995; Sibun and Reynar 1996), spelling correction (Zamora et al. 1981; Salton 1989), document categorization (Huffman and Damashek 1994; Labrou and Finin 1999), document comparison (Damashek 1995), robust handling of noisy (misspelled, OCR'ed etc.) texts (Grossman et al. 1995; Pearce and Nicholas 1996; Pearce and Miller 1997), topic highlighting (Cohen 1995), document space visualization (Fox et al. 1999; Huffman 1995; Charoenkitkarn et al. 1994), spoken document Retrieval (Ng and Zue 2000), and other information Retrieval related applications (Grossman 1994; Cavnar 1994). 5-grams have been most thoroughly investigated (e.g. Damashek 1995) and have emerged as an n-gram size capable of supporting even higher information Retrieval precision and recall than words as shown using the TREC-7 Ad Hoc task (Mayfield and McNamee 1998). Unfortunately, n-grams that are too long will fail to capture similarity between different but similar words, and n-grams that are too short will tend to find similarities between words that are due to factors other than semantic relatedness.

4.1.2 Problem

Despite the advantages of 5-gram based text processing, the number of different 5-grams creates its own challenges. There are $26^5 \approx 10^7$ possible sequences of 5 letters and roughly 50x more than that if other common characters (spaces, digits, punctuation) are included. To implement a table containing the number of occurrences of each possible 5-gram in a particular document in an array containing one entry per possible n-gram would thus require a very large array. Memory requirements have posed significant problems (Ng and Kantor,

1995), yet as memory becomes more available, the number of documents to process tends to increase as well. Thus, it would be useful to encode documents in terms of their 5-grams in a more memory-efficient manner (Crowder and Nicholas 1996).

Because most of the possible sequences of n characters rarely or never occur in practice for n=5, a table of the n-grams occurring in a given text tends to be sparse, with the majority of possible n-grams having a frequency of zero even for very large amounts of texts. For example, 40 MB of text from the Wall Street Journal were found to contain only $2.7*10^5$ different 5-grams out of a possible $7.5*10^{18}$ (based on an alphabet of 27 characters, Ebert et al. 1997). Even the entire Web is quite sparse (Table 4-1).

In Table 4-1 ten random letter 5-grams were generated, then searched for on the Web. Only two were found by the Alta Vista search engine (as of 8/2/00) anywhere on the entire Web in a non-binary document or its URL, in any human language. For a single document, a vastly lower percentage of all possible n-grams would be present.

Sparseness is further increased by the fact that the number of distinct n-grams in a given text is strictly limited by the document's length. A document containing M characters contains a maximum of $M-n+1 \approx M$ distinct n-grams, and normally contains fewer since some n-grams occur more than once.

Table 4-1. Some random 5-grams

Ten random letter 5-grams	**Number found on the Web**
obive thjfs jpomz aqzmk owsop znqfm xifiq mkxre zgwhb jclur	2 (jclur & obive)

4.1.3 Solution Approach

The sparseness of the n-gram table suggests compacting it. One way to reduce the size of the table of n-grams in a text is to simply store a list of the n-grams that actually occur, in association with the number of times each appears in the text. This provides an accurate accounting. Another way to reduce the table size is to hash the large space of possible n-grams into a considerably smaller array. This risks collisions, which may be duly recorded for accuracy or, alternatively, ignored for the sake of computational efficiency, in which case a given location in the array could contain a number expressing the summed occurrences of two (or more) n-grams.

The computational advantages of ignoring collisions are substantial. Even if only the 26 English letters are considered, there are still 26^5 potential 5-grams. Interpreting the bit pattern of each 5-gram as an integer without hashing leads to a sparse 256^5-element array for storing the n-gram content of any given document. Because of its sparseness, a great amount of memory can be saved by hashing, with tolerably few collisions. For example, with a hash table of size$\approx 2^{18}$, a document can be represented by an array with about 2^{18} elements, which is 0.000024% of 256^5. If a hash table of size$\approx 2^{25}$ is used, that size is still only 0.0031% of 256^5. As a result, the strategy of hashing 5-grams and ignoring any resulting collisions to support speed and simplicity has been found useful (Damashek 1995).

This paper investigates collisions in 5-gram hash tables, with the goal of minimizing their occurrence. We show that, using the commonly employed hash function *h(k)=k%tablesize* for prime *tablesize*, different table sizes exhibit wide variation in collision probability. We show that this pattern of variation is similar for the three languages investigated, English, French, and German, but not for randomly generated n-grams. In order

to support systems that use hash tables of 5-grams, we empirically determine and provide a list of table sizes covering a range from approximately 2^{16} to 2^{40} that have low collision probabilities. Choosing from the table sizes in this list avoids the possibility of inadvertently choosing a table size with an average or a high collision probability.

4.2 Methodology

We investigated the effect of hash table size on collision probability for 5-grams. To hash them, each was converted to an equivalent 5-byte integer i from 0 to 2^{40}-1, as follows:

Let the characters in the n-gram be named $a_0 \ldots a_4$, starting from the leftmost character.

Let $num(a_n)$ be the 1-byte integer equivalent of character a_n.

(1) $i = a_0 + 256a_1 + 256^2 a_2 + 256^3 a_3 + 256^4 a_4$

The resulting values were hashed using the standard hash function $h(i)=i\%tablesize$ where *tablesize* is prime. For each language tested, 100 documents from the Web were hand-picked and checked to ensure that each was written in the desired language and was a real document of reasonable length. This resulted in a total of 436,950 bytes of English, 469,638 bytes of German, and 446,032 of French. The English documents were from an ad hoc, diverse set of sources, while the French and German documents were from various university sites in those countries.

In order to determine when a collision occurred, the hash codes were first stored in an array in which one element stored the hash code of each n-gram in the file. The hash code of the kth n-gram in the document was stored at array index k. This required b-4 array elements for the b-4 n-grams in a file of b bytes. Then, the array was scanned for recurring hash codes. A recurring code could indicate a collision, if the two occurrences of the code derived from different n-grams, or a non-collision, if the two occurrences derived from different

occurrences of the same n-gram. This was determined by using the array index at which a hash code was stored as the offset into the document where the corresponding n-gram would be located.

4.3 Results and Discussion

We investigated a variety of table sizes between 2^15 to 2^40 for each of the three languages, as described next.

4.3.1 Collision Probabilities, Table Sizes and Languages

Different table sizes exhibit wide variation in collision probabilities (Figure 4-1). These variations are similar for English, French and German. Observe the *'s in the three overlapping graphs of Figure 4-1. Each * represents an average of the collision probabilities for the 20 prime table sizes nearest to but below a given power of 2. In all three graphs the pattern of *'s is generally descending, but exhibits certain upward exceptions, for example at 2^{24}. For all three languages, the average collision probability that occurred when hashing the n-grams in a document is substantially higher, on average, for table sizes just under 2^{24} than for table sizes just under 2^{23}, 2^{22}, and even 2^{21}. Similarly, upward exceptions to the generally descending trend occur at 2^{31} and 2^{32} for all three languages. This contrasts with a control condition in which a similarly size set of randomly generated n-grams were processed and graphed the same way (Figure 4-2).

To determine the statistical significance of these results, observe that for each language, the highest magnitude exceptions to the downward trend for the *'s are at the same three points out of the 25 plotted, 2^{24}, 2^{31}, and 2^{32}. Given that a language L_1 exhibits this property, we seek the probability of a statistical null hypothesis that both other languages L_2 and L_3, exhibit the same property due to chance.

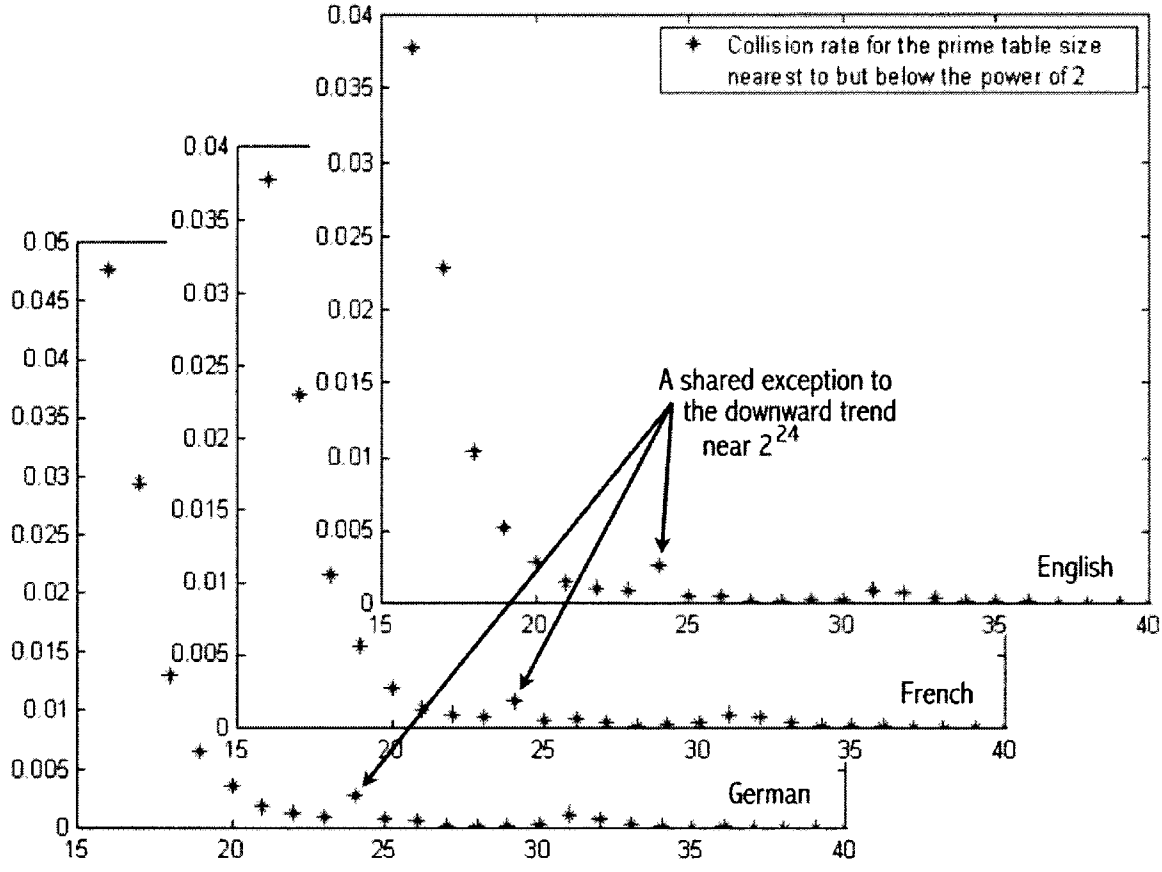


Figure 4-1. Graphs showing collision probabilities (y-axis) vs. hash table sizes (x-axis). Each plotted point is the average for the 20 largest prime table sizes below 2 to the powers 15 through 40. English, French, and German all show a trend of generally decreasing collision probability with increasing table size, consistent with well known properties of hash tables. Obvious upward exceptions to the downward trend are evident in the *'s for table sizes near the same powers of 2 for all three languages, though not for randomly generated n-grams (as shown in Figure 4-2).

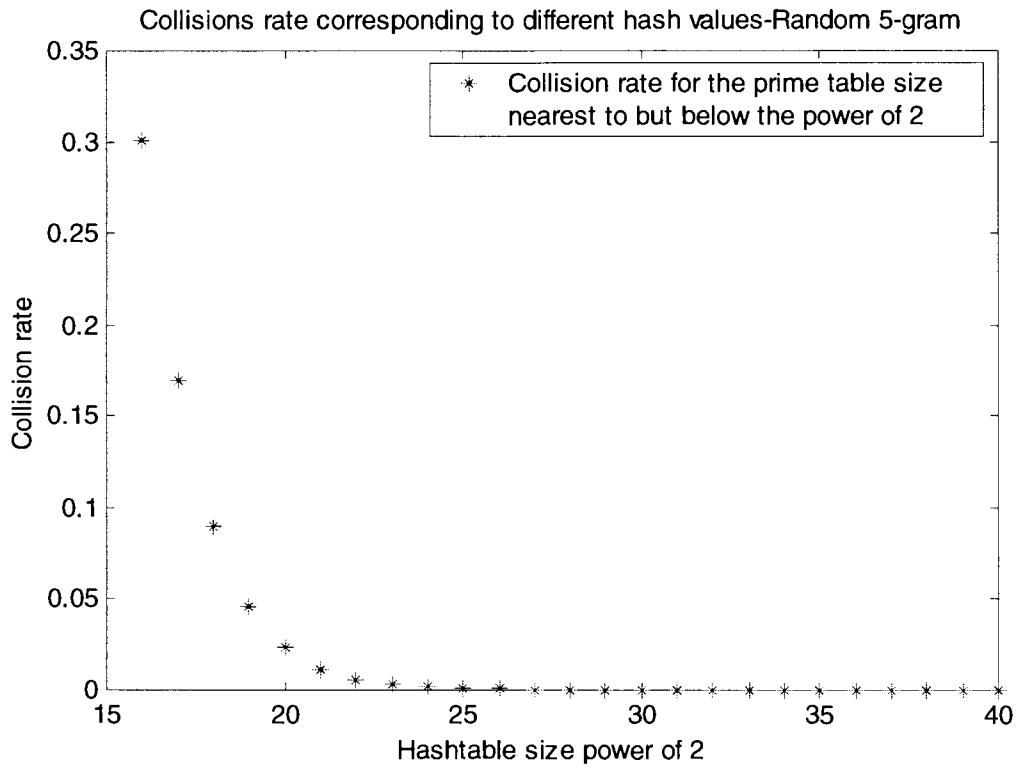


Figure 4-2. The collision probabilities corresponding to different hash table sizes for randomly generated 5-grams. This is the control condition, which we compare with the graphs for the English, French and German shown in Figure 4-1.

1) Assumption: L_2 and L_3 each have at least three exceptions to the downward trend. This assumption is conservative, therefore allowable, and is met in the current case. Call these exceptions E_1, E_2, and E_3.

2) $P(E_1$ is in the same place in the sequence of 25 plotted points as one of the three highest magnitude exceptions in $L_1)=3/25$. Remove that place from further consideration.

3) $P(E_2$ is in the same place in the sequence of the remaining 24 plotted points as one of the two remaining highest magnitude exceptions in $L_1)=2/24$. Remove that place from further consideration.

4) $P(E_3$ is in the same place in the sequence of the remaining 23 plotted points as the remaining highest magnitude exception in $L_1)=1/23$.

5) $P($all three highest magnitude exceptions in L_2 correspond to the exceptions in $L_1)=$ (3/25)*(2/24)*(1/23)=0.000435.

6) For the third language L_3, the analogous probability, $P($all three highest magnitude exceptions in L_3 correspond to the exceptions in L_1 and $L_2)$ is also 0.000435.

7) $P($both L_2 and L_3 have the same pattern of exceptions$)=0.000435^2=1.9*10^{-7}$ which effectively rules out the hypothesis that the results are due to chance.

4.3.2 Empirical Determination of Good Table Sizes

In order to support systems that use hash tables of 5-grams, we empirically investigated hash tables whose sizes are the 20 largest primes below powers of 2, for each power of 2 from 2^{16} to 2^{40}. From this we identified which of the 20 table sizes resulted in the lowest collision frequency, for each power of 2. Given a power of 2 representing the approximate table size desired, using the table size identified means both using a table size with a collision probability lower than for nearby table sizes, and also avoiding the possibility of inadvertently choosing a table size with a particularly high collision probability. Table 4-2 shows these table sizes together with the collision probability for each. To choose a good table size for a particular application, pick a table size from the "Best hash table" column. The other columns compare these table sizes with similar,

Table 4-2. Hash table sizes and collision probabilities. The best hash table size is the table size with the lowest collision probability of the 20 largest prime table sizes below 2 to the power indicated in the left hand column. The worst hash table size is the table size with the highest collision probability of the same 20 table sizes.

Power of 2	Best hash table size	Collision probability	Worst hash table size	Collision probability
16	65393	0.03453484380364	65407	0.04291337681657
17	130843	0.01656482435061	131071	0.06934431857192
18	262051	0.00804668726399	262139	0.01564938780181
19	524203	0.00379448449479	524287	0.01173131937293
20	1048423	0.00181714154938	1048571	0.00437807529466
21	2096957	0.00082847007667	2097143	0.00254262501430
22	4194181	0.00039134912461	4194301	0.00307586680398
23	8388283	0.00030667124385	8388427	0.00293855132166
24	16776931	0.00041423503833	16776961	0.02023343631995
25	33554239	0.00028836251287	33554347	0.00181485295800
26	67108747	0.00013502689095	67108859	0.00378761872068
27	134217493	0.00002517450509	134217467	0.00080787275432
28	268435091	0.00002517450509	268435331	0.00029522828699
29	536870791	0.00001373154823	536870909	0.00124957088912
30	1073741467	0.00002288591372	1073741419	0.00116947019110

Table 4-2. Continued

31	2147483249	0.00002288591372	2147483647	0.01076324522257
32	4294966769	0.00004348323607	4294967291	0.00602357249113
33	8589934289	0.00003890605332	8589934583	0.00220162489987
34	17179868809	0.00001830873098	17179869107	0.00038906053324
35	34359737821	0.00000228859137	34359738299	0.00028378533013
36	68719476619	0.0	68719476731	0.00070259755121
37	137438953331	0.0	137438953403	0.00007323492390
38	274877906813	0.0	274877906687	0.00003890605332
39	549755813869	0.0	549755813797	0.00001830873098
40	1099511627689	0.0	1099511627689	0.0

naïvely chosen table sizes and with the worst table sizes from the set of 20 from which the best table size was identified.

4.4 Conclusion

5-grams have been found useful for profiling documents in a number of text processing tasks. To best exploit the computational advantages that hash tables provide in storing the 5-gram profiles of documents, a common and simple hashing scheme should be used, and collisions should be ignored but their rate should be minimized by appropriate choice of a hash table size. We have shown that the properties of English, French, and German are similar for this purpose, and have empirically identified good hash table sizes over a wide range.

4.5 How to Run the Program

Software has been developed to do the experiment. For each language, a set of web documents has been downloaded to the local disk. For a certain hashtable size, the software reads each document into the memory, deletes the html tags, computes the total number of collided n-grams and saves the results in a data file. Then the data is analyzed using Matlab and the corresponding graphs are drawn to show the results.

5 CONCLUSION

How to provide users a more friendly user interface is always a very important issue in applications. This problem is especially important for situation such as when multiple documents are involved.

Multibrowser is built on the basis of direct display and multidisplay, which are more suitable in situations where users want to browse the contents of multiple documents. Before the documents are displayed, they have been processed based on their n-gram values, and the relationships among the documents are constructed according to their distribution in the document space. Each document is assigned a color bar according its position in the document space. Furthermore, documents are processed at the paragraph level, which provides fine-grained matching of related topics. Display is via a direct multidisplay HCI strategy to facilitate efficient interaction with texts. When documents are displayed on the screen, users can browse the content of several documents simultaneously and see the relatedness among documents by the color bars attached to the documents. The more similar the colors are, the more similar the documents are. In addition, users can easily find the most related paragraphs of a certain paragraph by clicking the "find similar" link following it.

Multibrowser provides a solution by combining document pre-processing and direct multidisplay.

APPENDIX. USING SOFTWARE ENGINEERING CONCEPTS TO ENHANCE LEARNING: A NOVEL APPROACH

Abstract

Teaching of many topics in computer and electrical engineering, engineering in general, and indeed many other fields appeal to software in some form. Some topics require students to use existing packages. Knowledge about testing and usability can help students understand limitations and get used to using existing software products. As a more technical example, teaching of algorithms for computer networks can benefit from a number of software engineering techniques for specifying what software does, because those techniques help to clearly describe those algorithms in a variety of ways. Such techniques can emphasize visualization (benefiting students who respond to the visual modality), logical sequences (benefiting sequential learners), summarizations (benefiting global learners), and others. The diverse techniques of software engineering can address problems arising in numerous contexts, both in courses and in engineering practice. The proposed approach can therefore serve students and instructors in a variety of courses, and the integration of software engineering into these courses can both help students to learn topics in those courses as well as become familiar with software engineering techniques through their use.

Software engineering is a large collection of diverse techniques – and overarching principles – that guide the entire process of producing software. This process starts with talking to clients before writing a single line in a computer language, to designing and writing software, to fixing problems and providing new capabilities in existing software.

Properly applied, software engineering leads to quality computer programs that do what users need them to do. Improperly applied, it leads to unending problems with the resulting software. Great potential exists for using software engineering concepts and techniques to enhance learning in a variety of important courses in the Department of Electrical and Computer Engineering.

Table of contents

1 Introduction

Software engineering has rapidly become a major topic area in computing education and its techniques have been widely adopted by companies and government agencies for software development projects. Departments of software engineering and degree programs in software engineering are becoming more common, and guidelines for software engineering education have received increasing attention. As part of this trend toward increased educational emphasis in software engineering, programs in computer science and engineering have been moving toward giving their students a significant foundation of knowledge in software engineering. We want to make software engineering a direct contributor to other areas of education that focus on non-software engineering educational goals. The result will benefit pedagogy throughout the curriculum, as well as better integrate software engineering into the curriculum.

2 Implementation

Several courses can be used to examine how software engineering concepts can help in other courses in the Department of Electrical and Computer Engineering. For each course that we examined, we identified a specific course topic that is important for students to comprehend and that will likely benefit from use of software engineering concepts. Typical suitable topics include those requiring teamwork, orderly thinking about synthesizing solutions to complex, ill-structured problems, communication with clients, specifications and design, and understanding of algorithms and other data manipulation procedures. For the chosen topics, we determined how specific concepts and techniques from the software engineering field can help in teaching the topic, and prepared a concise report explaining this.

The report was prepared with advice and/or other contributions from the instructor, and provided to the instructor.

For the Senior Design course, abstracts of students' proposed projects were screened to identify projects most likely to benefit from the proposed approach. The student teams for those projects received some consultation from their advisors and were required to use software engineering concepts in their work. At the conclusion of their projects, evaluations will be obtained from their advisor, from the faculty advisors, from the students themselves, and from the clients who suggested the projects as to the effects of the use of software engineering principles. For the Computing in Java course, team projects follow the software engineering life cycle with emphasis on the functional requirements. For the Computer Networking and Data Communications course, a communications protocol has been taught using the proposed approach. Ability to understand protocols is a core part of this type of course. Student learning of the protocol will be assessed using a homework assignment and a quiz. For all courses involved, instructor feedback will be obtained on both their satisfactions with the proposed approach and its utility.

3 Student Learning Enhancement

Current practice is to partition software engineering concepts into specific software engineering courses. Yet, such concepts can be used to help communicate diverse topics in many other courses. For example, software engineering can help teach students to address large, complex problems requiring teamwork. Another example appears in the Appendix. In industry and government, software engineering principles are typically used but practitioners generally learn this on the job unless they are young enough to have taken a formal course as students.

Students will better learn effective ways to visualize, communicate, document, decompose, and solve problems arising in courses and in the typical engineering environment of team development of complex systems.

4 Project Significance

Information Technology (IT) is currently a major initiative at the university level. The College of Engineering and the Department of Electrical and Computer Engineering are major supporters of this initiative. Software engineering is a key concept in IT. Additionally, a multidisciplinary initiative in software engineering at ISU is being developed to create a program to meet the high demand of regional industry for students trained in this area.

Using techniques from software engineering, instructors will be able to use them when appropriate; helping students to learn valuable techniques they can apply in their professional careers.

5 Ways to get project ideas

For each course, we have to collect materials about the content of the course, difficult spots, etc. There are several ways to get the relevant information.

Talk to past students of the class for possible trouble spots in the course

This could give misleading information since students don't always see the overall picture: where they have problems others might not, or vice versa.

Ask TA if they have noticed any areas where students do not understand the topic

This idea would work well for courses that are lab-based, especially if the professor points the lab out as a possible trouble spot. Also, the TA might have ideas of more trouble spots if many students come for help for specific areas or homework assignments.

If other professors have recently taught the course, they might have ideas of where SE techniques might be useful

This could cause coordination problems. But for classes that are switching professors, the past professor might have used SE previously, or know where problems might lie.

5.1 Approach the professor in one of following ways for ideas

Make sure the professor feels involved in the process, not like we are trying to tell them how to run their classroom – they are the teachers, not us. But hopefully we can offer some new ways of looking at things.

The following questions can be used to get some information from professors:

- Do you have initial thoughts about how SE techniques could help?

  This would give a good feel of what they expect, would assist in future dealings with the professor, and why they are involved in the project.

- What are some of the topics that students seem to find to be sticking points?

  This approach forces them not to think about the solution, which we are dealing with, but the problem that they might not be aware that we might be able to help with.

- Where do you think SE techniques would help?

  Gives us an idea of how much they know about SE. This would also be a good starting or ending place for a conversation.

- How do you think SE principles could be applied to your course?

  Would let us know what they are thinking, why they are involved in the project, and more about what they expect to get out of it.

- Ask professor if they already use any SE principles and/or techniques in their course.

  If so, which one(s), for what topic, and do they use SE for a project, just a homework assignment, for teaching the subject in the lecture, or another way?

  Possible ideas could be to supplement what they are already doing – we don't want to reinvent the wheel. But maybe there is another technique they could try that would do the same thing, in a better way.

- Ask professor for topic ideas where SE might help in their course.

5.2 Approach professor with solution ideas

Depending on the professor, and the class, this could be different for each class. Also depends on how many viable solutions there are for each topic, and how many different solution SE techniques we have for the class.

- Present SE techniques that can be used independent of topic.

  List SE techniques (solutions) (which could be used differently for each topic), and match each technique to the topic(s) it solves. This would be used to let the professor choose the SE technique they are most comfortable with, and then select the topic(s) that the SE technique could help teach.

- Address each problem separately, with a list of possible SE technique solutions

  The professor would choose the topics or problems they are most concerned with, or most interested in having SE techniques applied to, and then we (with or without the professor) could choose the SE technique that fits that area.

- Have a list of problems/topics

  For each, have one viable solution.

  This solution would be if we selected the best SE technique for each topic before approaching the professor with our ideas. The professor could choose from the topics, similar to the processes in both of above solutions.

  We know which one we would want to do, but have at least 3 options. Let the professor have most of the input in choosing which one. This is again letting the professors teach their own classes, not telling them which one we want to do. But know which area we are interested in.

  Possibly we could lean a little toward having different classes use different SE techniques. Then we would have a variety of SE techniques to analyze and evaluate their performance in aiding learning.

Examine the syllabus for ideas

We need to have an idea what we are doing before we approach the professor, so we look like we know what we are doing, and aren't just looking to them for the answers. If we already have an idea what the class is about, they won't feel like they are doing our job for us.

Brainstorm for ideas

This would be necessary both before and after approaching others for ideas, because as stated above, we need ideas before we approach the professor, and the professor's ideas could generate some new ones.

6 Case study

6.1 CprE 184X

6.1.1 Course background

This is a 2 credit course. About 30% of the students have little or no programming background. Lectures are 1 hour/week and labs are 2 hrs/week Homework emphasizes writing up lab reports and robots are used.

6.1.2 Possible software engineering approaches used in CprE 305

6.1.2.1 Possibility 1: Communicate Teamwork Issues

- Fact: 2-person teams seem to work well in the labs.
- Solution: a lab in which 2 separate 2-person teams must coordinate.
- Example: Make robot go to a destination over a given path.
  - Then make it return to the origination point.
  - Teams must coordinate to solve the problem.
- Instructor suggests that the robot's paths could be curved, and points out that the course has more robot labs than non-robot labs.

- Teamwork can be based around a *design*, or a set of *specifications*.
  - (Instructor prefers the design approach).

Design options are addressed next (specification options after that).

6.1.2.1.1 Possibility 1a (Teamwork): Design options

- Specifications are given by instructor (the "customer") and include.
  - Shape and length of path; time limits; resource limits (personnel, equipment, tools, hardware and software, etc.).
  - Evaluation criteria.
- Need for good communication can be based around a design document.
- Have teams cooperate and coordinate using the design document.
- A design document is produced be each team.
  - Design covers both hardware and software aspects.
  - Give them an outline so they know how to write a good one.
    - Outline includes e.g. technical approach, resources needed, etc. (to be fleshed out more later).

Four options for doing this are next **(instructor prefers option 2)**

- Option 1: team 1 develops first draft, team 2 reads, together they do a formal review, and iterate until done.
  - Both teams work as one larger team from the resulting document.
- Option 2: team n develops design, passes it to team n+1, team n+1 submits a grade for it, uses it to solve entire implementation task, then submits another grade for it.
  - Even teams write it, odd teams use it, OR

- Each team designs to slightly different specifications, and all teams design and implement.
- Some details: this exercise relies on a four part schedule:
  - Wednesday - discuss what designs are.
  - Friday (lab) - pass out the specs at the end of class, they then do design for HW.
  - Wednesday - answer any questions.
  - Friday (lab) - each team implements another team's design.

- Option 3: team n develops design, passes it to team n+1, team n+1 implements it, team n's grade incorporates team (n+1)'s implementation performance.
  - Teams must not have access to their own designs when implementing!
- Option 4: one team writes a design to get robot there; partner team uses it to write a design to get robot back; both teams implement their designs.
  - First team implements while 2nd writes, so the teams' activities are staggered.

6.1.2.1.2 Possibility 1b (Teamwork): Specifications Options

- Basic constraints are given by instructor (the "customer") and include.
  - Time limits; resource limits (personnel, equipment, tools, hardware and software, etc.).
  - Evaluation criteria.
  - Requirements would need to hit a balance between being too much like specs and not providing enough guidance - system is simple so it would be easy to say too much.
- Need for good communication is based around *specifications* document.

- Have teams cooperate and coordinate via the specifications.
- A specifications document is produced by two teams jointly.
  - Specs cover the state of the robot after it gets there and before it returns.
  - Specs also cover relevant API-like issues.
  - Give them an outline so they know how to write good specs.
- Option 1: team 1 develops first draft, team 2 reads, together they do a formal review, and iterate until done.
  - They then develop separate software designs & glue them into one software download.
- Option 2: team n develops specs, passes them to team n+1, team n+1 submits a grade for them, uses specs to do design and implementation, then submits another retrospective grade for them.
  - All teams write specs, and do a design and implementation.
- Option 3: team n develops specs, passes it to team n+1, team n+1 designs & implements, team n's grade incorporates team (n+1)'s performance.
  - Teams must not have access to their own specs, to get them to use another team's.

6.1.2.2 Possibility 2: Communicate the rapid increase in confusion as team size increases

- This can be demonstrated mathematically, with a hands-on exercise, or both.
- Exercise approach would be suitable as an in-class exercise; could also be done in a lab.
- Would make a classroom activity that is fun as well as educational.

- Three possibilities follow; one could be used or they could be combined (Instructor prefers possibility 2c).

6.1.2.2.1 Possibility 2a (Team size): Showing it Mathematically

- Show it mathematically, explaining the theory.
  - 1 person, 0 communication paths.
  - 2 people, 1 communication path.
  - 3 people, 3 communication paths.
  - 4 people, 6 communication paths.
  - 5 people, 10 communication paths.
  - 6 people, 15 communication paths.
  - 1+2+3+4+...N=about N*N/2.
- Give story about Euler's teacher telling him to add up all the numbers from 1 to 100.
- N*N means things get worse fast.
- Calculate how many paths for the number of people in the class (a lot!).

6.1.2.2.2 Possibility 2b (Divide Class into Ad Hoc Differently Sized Teams)

- Have different sized teams compete, showing an optimum size.
  - ...like the one I've used, maybe fine tune it.
  - Give each team a lengthy arithmetic problem like.
  - 3+6-1-10+8+2-7. . .
  - They are timed while they decompose the problem, solve it, and get the answer.
  - Typically small teams solve it faster than large teams!
  - Graph on the board the results, showing that bigger teams aren't usually faster.
  - Explain why.

- Could be combined with possibility 2a (to explain the principle involved).

6.1.2.2.3 Possibility 2c (Well and Poorly Organized Teams)

- Have largish teams, organized (or not) in different ways.
  - Have them compete; the well organized teams should do better!
  - "Well organized" implies the # of communication paths is controlled.
  - Assign different teams different organizational structures.
  - Follow-up discussion: people relate their experiences in different structures.
- Could be combined with possibility 2a (to explain principles).
- Could use the same task as in possibility 2b.

6.1.2.3 Possibility 3: Understanding and Using the engineering life cycle

- It's better to settle on specs before designing, and settle on a design before implementing.
- General idea: they run through the phases of the life cycle in one class session.
- Depending on the exercise, it could be done in either class or lab.
- Changes to specs could be made during design, implementation, and after implementation.
  - The point: illustrate that changes that could be made early, are much more expensive if made later.
- Instructor found this to be more suitable as an exercise to be done toward the end of the semester, but before the last larger lab project. It would be a fun thing to do in class.
- Outline of procedure.
  - Exercise is completed in one class or lab.

- Exercise requires them to
- . . . read, or write a spec document,
- . . . then a design document,
- . . . and *finally* they can implement.

6.1.2.3.1 Possibility 3: Task 1 - Spaghetti Towers

- Task 1: build towers of spaghetti and tape towers.
  - Each team gets spaghetti and tape.
  - "Best" tower wins.
  - Option 1 - give them the specs.
    - Tell them how much spaghetti and tape they get.
    - The winner is the tallest tower.
  - Option 2 - they make their own specs.
    - Provide them with an evaluation function.
    - They can choose how much spaghetti and tape to use.
    - . . . but using more impacts the evaluation function.
    - Evaluation function combines height with amount of spaghetti and tape.

6.1.2.3.2 Possibility 3: Task 2 - Paper and Tape Containers

- Task 2: build containers out of a piece of paper and tape.
  - Container that holds the most, using the least tape, wins.
  - (Need some formula).
  - In this case they read, not write, the spec document.
  - Would be fun to have some material (e.g. corn) that they use to measure container capacity,

- . . . or capacity could be measured with ruler and calculation.
- To show that changes should be made as early as possible:
  - Make a change in the specs after they've done the design,
  - ...and/or, make a change in the specs after implementation.
  - A possible such change would be a reduction in volume.
  - Discussion would center around the fact that difficulty of fixing a problem rises dramatically as it is delayed to later life cycle phases.
  - A house analogy fits the paper construction:
  - Easy to change around where the bathroom is during specs
  - ...harder to change the blueprints (i.e. the design)
  - ...most difficult and expensive to do after the house is built

6.1.2.3.3 Possibility 3: Task 3 - Index Card Houses

- Task 3: construct buildings out of index cards.
  - They must specify their own evaluation criteria,
  - . . . i.e. they generate their own specs.
  - They can tear the index cards so it's not the standard card house task.
  - Advantages:
    - They produce the spec document, not the instructor.
    - Task emphasizes creativity rather than competition.
  - Disadvantage: uses up a lot of index cards.

6.1.2.3.4 Possibility 3: Including the Maintenance Phase of the Life Cycle

- Maintenance is of major importance (67% of the total effort, in software domain!),
- . . . often maintenance is not handled much in typical courses (despite importance).

- To incorporate maintenance into the life cycle exercise:
  - After a tower, container, or building is constructed.
  - . . . make plans to change it!
  - What changes would be harder? Easier? Why?
  - Any changes should go through the life cycle in miniature.
  - Actually make one or more of the changes.
  - Experiences could be discussed in class.

6.1.2.4 Possibility 4: good problem decomposition

- Communicate what makes a decomposition good or not so good.
- To do this, describe the two major concepts involved:
  - Cohesion (relatedness within a module).
  - Coupling (relatedness among modules).
  - Have them help rate cohesion, coupling with cars, families, etc. - everyday things.
  - Have exercise involving decomposition of software or robot problem.
  - No implementation needed - this is a design issue.
- Assessment: instructor prefers possibilities 1-3, but rates this one as potentially good also.

6.1.2.5 Possibility 5: Improve Understanding of Pointers

- Problem: pointers are hard to understand.
- Solution A: use Java, which makes pointers easy.
- Solution B: a clear semi-graphical approach to describing them, perhaps from UML.
- Solution C: a "simplest possible" pointers programming exercise.
- Solution D: a robot programming task using the pointer concept.

- Example: robot goes to some location.
- It finds data or a pointer.
- If a pointer, it goes where the pointer says,
- . . . once it finds data, it does something with it.
- The physical layout of the robot's arena could resemble a graphical depiction of a C pointer explanation.

- Assessment: less desirable than possibilities 1-3. Instructor currently has an in-class exercise in which students act out a computer, some as parts of the computer and others help execute it. A sorting exercise can show how changing pointer values can be more efficient (less movement of data/people) than a non-pointer solution.

6.2 CprE 305. Computer System Organization and Design

6.2.1 Course background

The course is taught using the Verilog HDL hardware description language. In general, it would be good to keep the students informed about why they are going through the process they are going. The course does things in accordance with good methodology but because it tends to be planned out for them ahead of time, thereby increasing their efficiency, they would benefit from better awareness of the ways in which what they do conforms to good methodology.

6.2.2 Possible software engineering approaches used in CprE 305

6.2.2.1 4GL features

Since the students use Verilog, a 4GL with C-like syntax and visual programming capabilities, problems typical of 4GLs, and of visual programming languages will be likely to occur. In particular, students may be likely to think that once a model is running, it is

therefore valid. With 4GL's, the assumption that once the program runs it is running properly is often a problem. By teaching the students to watch out for this temptation by checking the outputs of their programs for reasonableness, always a good idea in software development but easy to forget when using 4GLs, the products of their efforts would hopefully be more likely to be reasonable than they would otherwise be.

6.2.2.2 Parallel Statements

Verilog has some semantic differences from C that can be confusing. For example, a=b; b=c; can mean different things depending on whether the statements are executed in parallel or not, whether they refer to the actions of circuit components like flip flops that may be edge triggered or level triggered (adding different timing issues to the semantics), and whether they refer to different points along a net that is subjected to rapidly changing input signals.

Students can be confused by this and mistakenly assume the wrong semantics applies in a particular case.

One way to address this is to show students different semantic descriptions of the same code segment, such as a=b; c=d;. However this would be using concepts from compilers or programming languages more than software engineering. Another drawback is that some students may not respond positively to using a formal description approach as a method of clarification.

Another way to address this would be to describe a system such as in the figure using a UML construct such as activity diagrams, thereby illustrating the parallelism. Or perhaps better, have the students generate the diagram.

6.2.2.3 Bottom-up design

The students build their systems using a bottom-up strategy, while the design is predetermined. This could be motivated with a discussion appealing to the advantages and disadvantages of top down and bottom up design, in general and in the context of the hardware problem dealt with in the course.

6.2.2.4 Cohesion and coupling

System design requires decomposition. Could software engineering provide guidance in this? Perhaps the concepts of cohesion and coupling, along with a design strategy that encourages high cohesion and low coupling, would be helpful. In this course, the basic design of the system they spend the semester on is predetermined. They simulate NAND gates, then use them to do flip flops, use them to do registers, use MUXs, decoders, and registers to do register files, use those, an ALU which they do, and a memory module which is given them, and do a processor with those. This can make them think design is too easy, or "just happens." However, the concepts of cohesion and coupling could be used to help them understand the way the design is the way it is. Cohesion and coupling can be appealed to based on their intuitive meanings, or the more formal classification of modules into categories of cohesion and coupling can be used.

To rate cohesion of a module, its proper classification in the spectrum of cohesion categories should be determined. This spectrum of categories (from best to worst) consists of functional cohesion, informational cohesion, communicational cohesion, procedural cohesion, temporal cohesion, logical cohesion and coincidental cohesion.

To rate coupling among modules, its proper classification in the spectrum of coupling categories should be determined. This spectrum of categories (from best to worst) consists of data coupling, stamp coupling, control coupling, common coupling and content coupling.

6.2.2.5 Parallelism

Due to the parallelism that is typical of digital circuit operation, a way to help students visualize parallelism could be helpful. What might software engineering contribute to this need? Perhaps activity diagrams could be used to express parallelism. Petri nets can be used to describe the concurrent systems. Especially, it can be used for describing concurrent interrelated activities. However, Petri nets may be outside the desired scope of the course. Sequence diagrams may not be suitable for parallel system description. Scenarios may have too many cases to be suitable. So let us focus on activity diagrams.

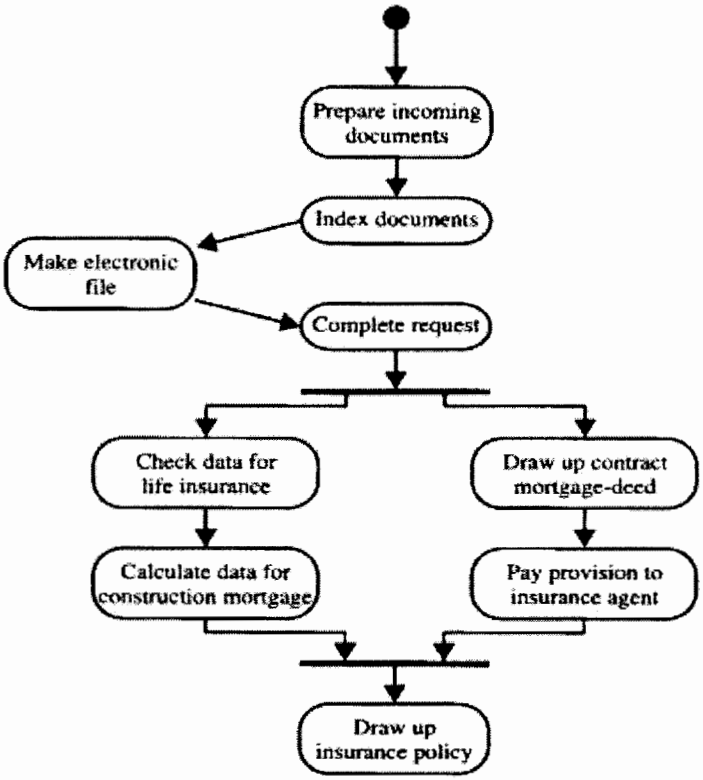
Activity diagrams. An activity diagrams is a state machine-like construct focusing on activities, which are states in which some computational activity is taking place. Activity diagrams show these activities, relationships among them, and the possible state transitions depending on different conditions. Heavy bars with more than one incoming or outgoing arrow indicate the start or end of concurrent processing. Activity diagrams are used to model sequences of actions and conditions within a process. They are like flow charts in important ways, but differ in supporting parallel activities and their synchronization. See figure A-1.

A potentially significant problem with activity diagrams as described is that they do not indicate which modules execute which activities and the way that the modules communicate. One solution to this is to organize actions into "swimlanes." A swimlane represents responsibility for part of the overall activity and may be implemented by one or more modules. Swimlanes are separated by vertical solid lines, as shown in figure A-2.

A related description method is the state diagram, which is similar to the activity diagram but does not stress states as places where activities are occurring. This is another option that could be explored.

6.2.2.6 Code Reuse Issue

A component is often designed, then used in different places in the system. This is a kind of code reuse. Students in this course do make use of reusability in that they build components that are then used in different places in their systems. However, the concept of reusability and the implications of building a component intended for reuse as opposed to one build without reuse in mind is never explained to them. Potentially such an understanding would be a good investment.



RE 5–20
tivity diagram for processing mortgage requests (Loan: Processing Mortgage Request).

Figure A-1: Activity diagram, quoted from Bahrami (1999) Figure 5-20.

- A reusable component can be a module, code fragment, design, part of a manual, or set of test data.
- Two types of reuse are accidental reuse, which means the developer accidentally realized parts of some existing products can be reused in a new product, and deliberate reuse, which means the components are constructed specifically for the future reuse.

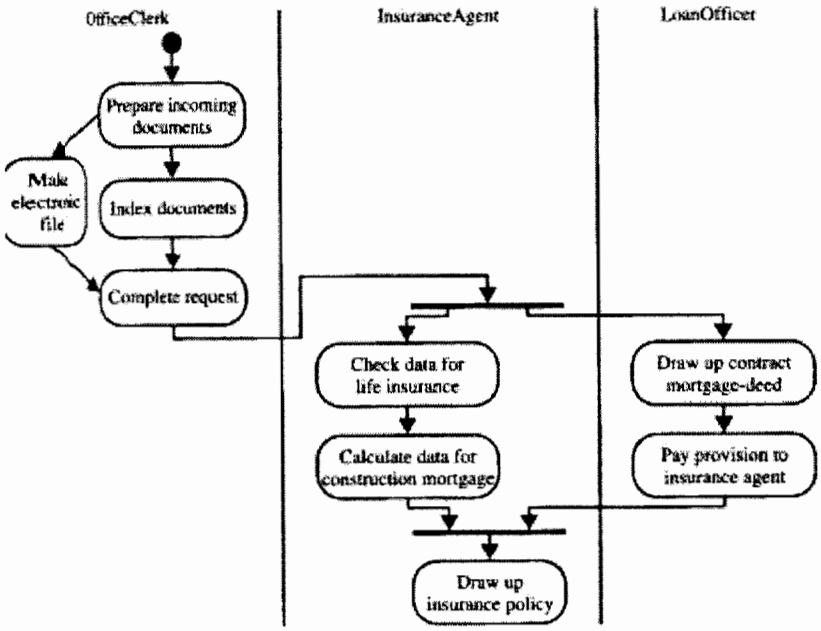


FIGURE 5-22
Swimlanes in an activity diagram.

Figure A-2: Swimlanes in an activity diagram, quoted from Bahrami (1999)

Here are a few facts about reusability.

- The reasons we need reusable components include: first, it can save much time because there is no need to develop parts of the new products; second, the product is more reliable because the reused components have been (or should have been) tested thoroughly, more thoroughly than if they were not for reuse.

- It takes more effort to design a component with good reusability than the equivalent one-time component. Components which are very likely to be used elsewhere should be constructed with reuse in mind. Components not likely to be used elsewhere can neglect the reusability issue. Object-oriented design can be used to design a product with good reusability.

6.2.2.7 Large System Development Issues

Large system development is an issue that could be addressed. How to develop large systems well? If there is a temptation to fix problems later rather than as early in the process as possible, then teaching about the relatively greater expenses associated with procrastinating fixes until later in the engineering process might be helpful. See figure.

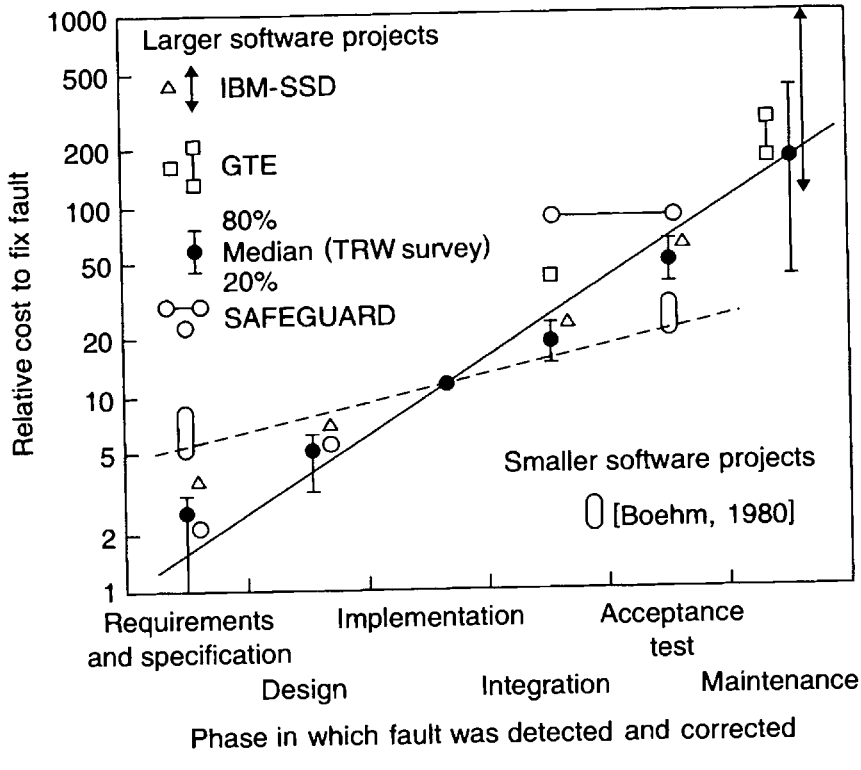


Figure 1.4 Relative cost of fixing a fault at each phase of the software life cycle. The solid line is the best fit for the data relating to the larger software projects, and the dashed line is the best fit for the smaller software projects. (Barry Boehm, *Software Engineering Economics*, © 1981, p. 40. Adapted by permission of Prentice Hall, Inc., Englewood Cliffs, NJ.)

Figure A-3: Software develop and its cost, quoted from Schach (1999)

6.2.3 Material developed for CprE 305

- Transparency 1:

Hello! Today's discussion is on:

How software engineering can help CprE 305 students

How does software engineering affect CprE 305?

You are designing hardware, not software…

…but you design hardware by writing software!

Therefore guidance on software development…

…applies to designing hardware with Verilog

By following good software development practice…

…you can develop software faster and better

The point of software engineering is to…

…develop software faster and better

The point of today is to help you to…

…develop your term projects faster and better

- Transparency 2:

So, what is software engineering?

An approach to software development

Based on the "software life cycle" idea

Better than the "no approach" approach

Becomes more important for larger projects

Provides good code structure

Provides good team coordination

For 1-week HWs, you don't need it!

For larger projects, you do

Industry emphasizes large projects

Few 1-week HW style tasks in industry

- Transparency 3:

How to do software engineering

- just follow a reasonable software life cycle model

...there's a bunch

...they're all the same

...lets take a look at some...

Now, let's look at resource allocation

Seems like "minor" phases just take a few %, right?

...but those few % are important

- as shown by the figure

- Transparency 4:

Let's skimp, and see what happens

Given: 9 week term project

...no maintenance (rightly or wrongly)

How many hours needed for each phase?

S'pose we skimp on requirements by 25%

...then make it up in specs, or design, or etc.

Let's make a table:

Phase	*Old time for phase*	*New time if fixed in*			
		Specs	*Des*	*Imp*	*Int*
Requirements					
Specs					
Design					
Implementation					
Integration					
TOTAL TIMES					

If time, let's make pie charts for each result column

- Transparency 5:

  Unexpected need for more time in a phase

  Given: you find a project needs an extra day of effort in phase P

  Option 1: you give it the extra day

  Option 2: you don't, fixing it later in implementation

6.3 CprE 308. Software Systems Integration

Should code inspections or code walkthroughs be used? In general, their "fault-detecting power...leads to rapid...early fault detection. The additional time required...is more than repaid by increased productivity" (Schach 1998 p. 462.). A relevant source is IEEE standard

1028 (1988), "IEEE Standard for Software Reviews and Audits." If an inspection or walkthrough is to be performed in class prior to implementing the code in lab, then the main question is whether to use the walkthrough technique or the inspection technique. Compared to walkthroughs, inspections are more formal and have more steps. Here are a few other relevant characteristics of each; modifications appropriate to this context will be needed and some possible ones are noted.

6.3.1 Walkthroughs

1) *Team has 4-6 members.* For purposes of the course, the team should have the same number of people as the lab teams, though not necessarily the same actual individuals.

2) *Team includes someone involved in writing the code, someone who will work with it later, the team leader of the team that wrote the code, an end user, and an SQA person.* For purposes of the course, this could be the team leader of the team writing the code, another member of that team, a member of the team who will extend it later (if possible, otherwise someone on another team), and someone from still another team taking the role of an eventual user of the system.

3) *Walkthrough chair should be the SQA person.* Actually, the chair should be the person who has the most to lose or gain from the quality of the result, so for the course, the chair should perhaps be either the representative of the next team to continue development of the software, or the team leader of the team that will implement the code in the next lab.

4) *The code is distributed ahead of time to the walkthrough participants.* For this course, this would be prior to the class meeting of the walkthrough.

5) *Walkthrough participants review the code and write down a list of questions and another list of probable bugs.* For the course, prior to class each participant should develop those two lists.

6) *Faults found during the walkthrough are not fixed, only found.* For the course, students should avoid trying to figure out what the fix is after identifying a fault. This saves time since the walkthrough must be completed during class and also, a single debugger will work more efficiently than a committee discussion.

7) *The walkthrough may be driven by going through the lists, or by going through the code with interruptions as needed when the current location in the code triggers a list item.* For the course, time may be a factor. List driven walkthroughs are faster, but less thorough. Which would be best for the course?

8) *Walkthrough results are not used for personnel evaluations, to avoid motivating any participants to be defensive or hold back on noting problems.* For the course, this simply implies that the walkthrough itself not count for a grade, at least not beyond simple participation or not.

6.3.2 Inspections:

1) An overview of the code precedes the individuals' reviews, which precedes the main inspection meeting. This suggests two meetings, one for an author to overview the code for the inspection team, and another for the actual inspection.

2) A checklist of possible faults to look out for is provided, and the inspection session is driven by methodically going through the code. For the course, the instructor would provide the checklist of faults.

3) A written report on the inspection is produced within 1 day of the inspection session. This would need to be done by the inspection team leader. The one day deadline may not be feasible in the context of a course, but if due by email to the other team members, it is potentially feasible.

4) The code is reworked. This is prior to actually compiling it. For the course, one person would have to be responsible for the rework, perhaps meaning that someone else would then be responsible for the compilation step.

5) There is a followup stage, where the inspection session moderator makes sure all problems have been addressed by the rework. Again, this is prior to compilation.

6) Statistics are maintained on the number, severity, and kind of faults found. This is useful in detecting when something unusual is occurring, however in the context of the course, this step may not be necessary.

7) Inspection teams consist of 4 people, a moderator, a representative of the team that produced the product under inspection, a representative of the team that will extend the product under inspection, and a tester, and of those, someone must be responsible for recording the findings of the inspection. For the course, the tester could be someone from the team developing the software who will have responsibility for testing it.

8) *Results are not used for personnel evaluations, to avoid motivating any participants to be defensive or hold back on noting problems.* For the course, this simply implies that the walkthrough itself not count for a grade, at least not beyond simple participation or not.

9) faults found during the walkthrough are not fixed, only found. For the course, students should avoid trying to figure out what the fix is after identifying a fault. This saves time since

the walkthrough must be completed during class and also, a single debugger will work more efficiently than a committee discussion.

6.3.3 A third option

If we abstract out the "essential component" of both walkthrough sand inspections, it is that code is best written first, then debugged by visual examination, and only then is it compiled. (Research has shown this to be more efficient than compiling it immediately!)

A streamlined approach to exercising them on this basic concept not only may have more impact, but would be easier to integrate into lab exercises throughout the semester, should you choose to do that later, based on the one exercise that we're planning on for fall.

- The protocol then would be:
  - They listen to the appropriate lecture material in class, Student M then writes the code, Student N takes it home, reads it, and lists bugs and etc. They meet in lab, discuss the list, ...and proceed to compile and debug it until it works.
  - Then, those that follow the directions will reap the benefits, those that don't will continue to struggle during the lab sessions, and the only difference in lectures is the possible need to schedule things so they have time to hear the concepts, then write the code, then take it home and study it, and only then show up for lab.
  - A highly tuned, detailed learning experience on inspections would be nice, but the streamlined approach above may work better in practice.
- **Introducing students to the process.** The process should be explained during a lecture. The best time to do this is about 1/3 of the way through the semester. This can include the following possibilities:

- Hand out some code, have them look it over and try to identify bugs, for one of the classes.

- An-class exercise that runs through the process in miniature. In pairs, one of them can write a small program, give it to the other to study, that person makes a list of bugs or questions, the writer then steps the reader through and they discuss bugs, then they fix and hand in.

6.3.4 Material developed for CprE 308----"Code Reviews: Making Code Development More Efficient"

- Transparency 1:

More Efficient Software Development

- . . .with nonexecution-based testing
- Execution-based testing:
  - o try running it
  - o debug from there

Nonexecution-based testing

- Review the code without a computer first!

About Nonexecution-Based Testing

- Reviewing the code without a computer
- "Doesn't this waste time?"
- OS development:
  - o reviews decreased bug detection costs by 85%
  - o Switching system development:
    - § decrease was 90%

- o Jet Propulsion Lab saved $25,000 per inspection

OS development:

- § Reviews decreased bug detection costs by 85%

- (Source: S. Schach, Classical and Object-Oriented Software Engineering with UML and Java, 4th ed., p. 141)

- Transparency 2:

Should Code Reviews be Used?

- Yes:
  - o . . . they make software development more efficient
  - o You might think they waste time. . .
  - o . . . but apparently they save time!
- Let's do a code review next
  - o . . . so you can use it with, say, lab programs or term projects
  - o Reason: so you know how to develop code more efficiently

- In class exercise:

<u>Code Reviews: an Approach to More Efficient Software Development</u>

Today we will run through a code review process in miniature. You can then apply this idea in labs, team projects and other future software situations. Please hand in your results. To do this code review, simply follow the following steps.

1) On paper, write a C program. Leave plenty of white space so you can fix bugs later! It should solve the problem of counting up the month's heating degree days, and also the month's cooling degree days. Here's how to calculate the "heating degree days" for a given day: if the day's average temperature is 65 degrees or higher, there are no heating degree

days for that day because the building need not be heated. However if the average temperature is below 65 degrees then the number of degrees it needs to be heated to get up to 65 is the number of heating degree days for that day. To find the cooling degree days, compute the number of degrees of cooling needed to get the average temperature down to 65. The average temperature is simply midway between the day's high and low temperatures. Example: the high temperature on one day is 55 degrees, and low is 35 degrees. The average is then 45 degrees. This is 20 degrees below 65, so there are 20 degree days for that day. If that happened 30 days in a row, the cumulative degree days would be 30x20=600. Write a function that takes as an argument two arrays of integers, called highs and lows. Each array has one element for each day of a month, plus an extra element, highs[0] and lows[0], saying how many days the month has. highs contains the high temperature for each day, and lows contains the low temperature for each day. Step through the arrays, and calculate the average temperature for each day and use that to help add up the total "heating degree days," used as a measure of fuel use for heating on cold days.

2) Raise your hand. I will assign you to a team of 2.

3) Each team member reads the other member's code, writing down on it a list of bugs or questions.

4) Each team then discusses the bugs and questions.

5) Each person then independently fixes bugs in his/her own code.

6) Write your names on both programs and hand in.

7) Leave. As you leave, think: "code reviews make software development more efficient."

Here are examples of annual average heating and laces around the U.S.

Typical heating and cooling degree days for some cities in the US (source: USA Today)

Barrow, Alaska	
Heating degree days	20,370
Cooling degree days	0
Bismarck, N.D.	
Heating degree days	8,932
Cooling degree days	499
Hilo, Hawaii	
Heating degree days	0
Cooling degree days	3,134
Kansas City, Mo.	
Heating degree days	5,326
Cooling degree days	1,388
Key West, Fla.	
Heating degree days	68
Cooling degree days	4,820
Yuma, Ariz.	
Heating degree days	983
Cooling degree days	4,244

6.4 EE 424. Digital Signal Processing

Professor Julie Dickerson indicated that the most efficient and effective way to apply software engineering techniques in this course is as follows: "I think that the students have the most trouble in the lab trying to structure and document their code. It would be a good place to apply it." Consequently, we generated the following ways for students to structure and document in labs.

6.4.1 Student coding guideline----handout to the students

Good coding guidelines. Observing these simple rules for documenting and structuring code will help you work more effectively and efficiently – in this course as well as in future coding situations.

- **Documenting your code:**

It is important in EE 424 labs to document the code 'as you go,' so that the code is more easily understood when reused in later labs by someone else or by yourselves. Here is how to document your code for this purpose.

a. *Self-documentation.* Use meaningful variable and function names. A variable called 'n' doesn't carry any meaning of what 'n' represents, whereas if 'n' had been named 'filterMultiplier', the code will be that much more understandable. Another example of this is a function called 'filter'. Reusing this code in future labs could get difficult, especially when combining multiple filters together to create a more complex filter. Using a more descriptive function name such as 'EchoFilter' allows for simpler understanding and reuse. Good variable and function names are classified as self-documentation because the names themselves document their meanings.

b. *Leave a maintenance trail.* It is important to leave comments marking any code that you change, either by adding something or modifying it. Use your name and the date of your change. Then later, you will be able to more easily find possible places of error, and when you use the code for another lab, it will be obvious what part of the file is actually the part you wrote and want to reuse. This also serves to make less work for the TA, who will consequently be happier.

c. *Comments*. Comment almost everything you write. Comments should explain what the code is doing. For this reason, every line of your code should be commented unless there is an obvious reason that it does not need to be. For example, every loop can be commented at both the beginning and end. At the beginning, say what the loop does, and at the end say what loop is ending. Individual lines of code can be commented, stating briefly something that enhances the understandability of what the code is doing. Sections of code should also be prefaced with comments explaining what the code does, its inputs and outputs, programmer name, date, and anything else to enhance its understandability and future reusability.

- Structuring your code:

Programs should have good structure so that they can be more easily debugged and modified, and so that chunks of code can be more easily reused in later labs. Here's how.

a. *Indentation*. Good indentation has been shown to make code more understandable. Indent the statements in a loop, the statements in a function, or any time a change in margin will enhance understandability. When you indent, how many spaces should you use? One study showed that 2 or 4 spaces were best, but didn't test 3. It seems likely that the "top of the curve" was at 3 spaces, so we recommend indenting 3 spaces. That means any given statement starts 0,3,6,etc. spaces from the left.

b. *Blank space*. Adding a blank line between sections of code makes it easier to read the program. Usually a single line is sufficient. However, do not put in too much blank space, because that tends to make the code spread over more screens, requiring more scrolling around and making it more difficult to find a given section of code or get a birds-eye perspective on it.

c. *Copy or call?* A section of code should not be copied and reused in the same program. Instead, make it into a function and call the function in each place you want to use the code. This will allow you to debug more easily, since each fix won't have to be made separately to different copies. If each call should be slightly different (say a different value of variable *frequency*, you can define a function that takes *frequency* as an input argument. Another benefit of using functions is that each call is only one line, instead of several lines of repeated code. This makes the program easier to understand.

6.5 CprE/EE 465. VLSI

One reason why this kind of course is suitable is that it involves specification and verification problems in VLSI, and these have significant similarities to similar problems in software engineering. An interesting difference is related to design – for VLSI the physical layout encourages communication links to be with physically neighboring modules, whereas in software engineering the communication constraints are less. From a software engineering standpoint, the concepts of cohesion and coupling seem likely as candidates for communicating, explaining, and understanding some relevant design issues. Professor Akhilesh Tyagi notes that many of the people doing research in the area have been involved in somewhat similar work in software engineering.

The course is becoming a high-level design course. HDLs are indispensable – they change a hardware problem into a software problem. The course uses Verilog. Students' biggest problems tend to be in the labs. Some lab problems that confront students:

- Design tools are not user friendly enough. They crash themselves and their computers a lot. This makes labs frustrating. They waste lots of time and it's hard to get

through labs. Good programmers have fewer problems with Verilog. Some of these problems can change from generation to generation of the design software.

- Students often do things like specifying delayless loops. This causes system to have a slow response time due to infinite looping.
- It is easy to describe circuits that are impossible to build, and not realize it.
- Due to use of verilog, an object oriented design approach would likely be useful.

For debugging, the following possibilities present themselves as potentially useful here.

6.5.1 Possible solutions

6.5.1.1 Sandwich approach

Implementation strategy might benefit from the sandwich approach, which judiciously combines top down and bottom up implementation strategies, incorporating the advantages of both. Control mechanisms, which tend to be at the top of the tree-like graph describing which modules call which other modules, are implemented top down, and operational modules, which realize specific functionalities and tend to be at the bottom of the tree, are implemented bottom up, meeting in the middle.

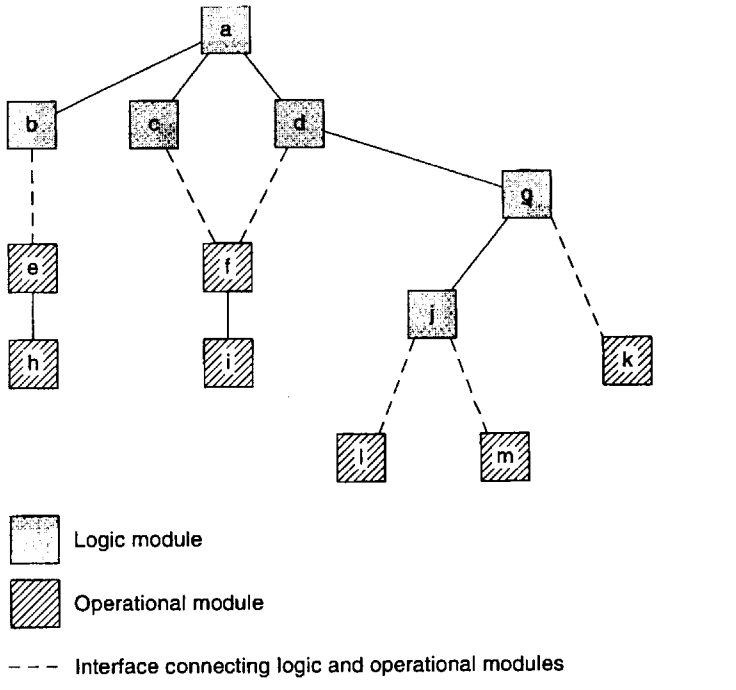


Figure 14.2 Product of Figure 14.1 developed using sandwich implementation and integration.

Figure A-4: Sandwich approach, quoted from Schach 1999.

The sandwich approach avoids a shortcoming of the top-down approach in that, while serious design faults tend to be detected early since they tend to be in the top level modules, the lower level operational modules may be inadequately tested, especially if they are to be reused later. The sandwich approach also avoids a shortcoming of the bottom up approach in that, while the lower modules are implemented early and therefore tend to be relatively thoroughly tested because they are run repeatedly while developing the upper level modules, serious design faults in the upper control modules tend to be detected late, leading to the risk of requiring significant redesign after significant implementation has been done.

Modules a, b, c, d, j, g are control modules, indicating they should be implemented top

down. Modules e, f, h, i, l, m, k are operational modules, indicating they should be implemented bottom up.

Some other design issues: the design process can first involve developing a system description at the behavioral level and simulating it. After verifying the behavioral description of the system by simulation, the system can be divided into several subsystems. This process will be performed recursively until all subsystems can be found in libraries or previous designs. Because in every step, the simulation of the behavioral description and the simulation of the hardware description are compared to verify the hardware description, it will be relatively straightforward to locate errors in the hardware description.

6.5.1.2 Hardware or software for implementation

A major design issue is deciding whether to implement something in hardware or software. What software principles could help in guiding such decisions? Dimensions such as speed, power, maturity of the design, and reusability all factor into such a decision. The requirements phase is the first step of software engineering life cycle. Only after the client's basic requirements are clear to the developers can the management plan, detailed specifications, design, and implementation proceed. Consequently, developers should decide whether to implement something in hardware or software in the requirement step. In requirements phase, the client and developers communicate basic characteristics of the system to be built. The developers can then make decisions about what to implement in software and what in hardware. The following factors can be used to make this decision:

1. Speed. The developers should determine if software can meet the need. If not, some parts of the system have to be implemented in hardware.

2. Maturity of the design. If a technique is not mature, it better to implement the technique in software because it easier to change than hardware.

3. Reusability. It is harder to build something for reusability than not. It is also harder to build equivalent functionality in hardware than software, enough harder that going the extra mile to make the hardware reusable in other contexts is worth serious consideration, even if a software solution would not be made reusable.

4. Power. Hardware consumes more power than software. If minimal power is required, software should be considered.

5. There may be other factors which may be required by a client. The developers should consider all these factors in deciding what should be done in hardware and what in software.

The software engineering approach of Structured Systems Analysis may be useful in this.

6.5.1.3 Debugging issues

Debugging is a process which involves locating and removing the errors. Errors can be divided into two groups: syntax errors and Logic errors. Syntax errors are caused by incorrect use of programming language and the compile can identify these syntax errors. So it's easy to remove. Logic errors mean that the program can be compiled and the executable file is generated, but output of the program doesn't satisfy requirement specifications. Logic errors are more difficult to be found and removed.

Normally, debugging process includes following phases:

- Information Gathering

In this phase, a programmer should collect as much information about the failures as possible.

- Fault Isolation

The aim of this step is to isolate the locations of the failures.

- Binary Partition Approach

If the program executes correctly until statement 1 and produces an incorrectly output in statement 2, it's obvious that the failure must be between statement 1 and statement 2. Then some methods, such as trace or a few output statements can be added at a point about the midway between statement 1 and statement 2. According to the output of this point, the failure can be restricted in the first half if the output of this point is incorrect; or the failure must be in the second half if the output is correct. This process can be proceeded recursively until the failure is identified.

- Structured Question and Answer Approach

In this approach, a series of structured questions are posed and answered to identify the failure.

- Involving others Approach

In this approach, new people are involved. These new people read parts of the Software Requirement Specifications and Software Design Specifications and then use binary partition approach to identify the errors.

- New Test Case Approach

If it is difficult to locate the faults based on the available test data reports; the fundamentally different test cases must be designed to gather more information.

It seems to me that the structured Question and Answer Approach and Involving Others Approach is more suitable in a very complicated software development process and need much more time.

New Test Case Approach can be used sometimes, but it maybe not very useful in class because it's not easy to construct fundamentally different test cases for most in-class programs.

Binary Partition Approach is a useful approach in everywhere from a very large system to a several-line program. It's easy to implement and it's really useful according to my experience.

- Fault Confirmation

After the fault location is isolated, a programmer has to make sure that this place is the sole cause of the failure.

- Documentation

The abnormal behavior of the program, test environment, correction and modification must be carefully documented.

- Fix the Fault

When the exact location of faults are identified and confirmed, remedial action should begin. Before correction is implemented, its effect on other place has to be evaluated carefully. And after the correction, a subset of the software, which includes the corrected part, has to be tested both locally to make sure that the correct doesn't affect other parts' functionalities.

- New Testing after Each Correction

After every correction, the software has to be re-tested. Especially, some parts which are related to the corrected parts have to be examined carefully.

Specific ideas on using software engineering ideas to enhance learning in 465.

- A lab in which they debug problem programs, geared to familiarizing them with the more troublesome bugs up front, so they can deal with them more easily when the occur later.
- An arrangement whereby students who identify bugs in the design and simulation software are rewarded, and the information is rapidly propagated to the other students, so that these bugs are worked around more efficiently, on average.

6.5.2 Online bug report system

This Web based reporting system allows students to fill out a form, submit it to a repository of submittals, view a table of contents listing all submittals and the date each was submitted, and click to view any one desired. (Figure A-5.)

Here are the components of the system:

- process.cpp is the CGI file which should be placed under the directory "cgi-bin".
- reportForm.htm, body.html, mainpage.html, title.html, and no_record should be placed under the directory "htdocs".
- A subdirectory named "body", which is used to store the records separately, should be created under htdocs.

Title: brief description of problem:

Details of problem:

Severity of the problem (check one)
- minor
- minor-to-moderate
- moderate
- moderate-to-severe
- severe

What do you know about the cause of the problem?

How was the problem handled?

Did the problem prevent completion of the lab (check one)
- Yes
- No
- In part

Recommendations for other students

Submit Clear

Figure A-5: An example display of the bug report system.

6.5.3 Source code of this CGI program

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream.h>

#define MAX_ENTRIES 50
#define LF 10
#define CR 13

typedef struct {
  char *name;
  char *val;
 }entry;

char *makeword(char *line, char stop);
char *fmakeword(FILE *f, char stop, int *len);
char x2c(char *what);
void unescape_url(char *url);
```

```
void plustospace(char *str);

FILE* title, *body, *recordNo;

//ofstream title("title");
//ofstream body("body");

void main(int argc, char *argv[]) {
  entry entries[MAX_ENTRIES];
  register int x,m=0;
  int cl;
  int no_record;
  char* fileName=new char[20];
  char* fileUrl=new char[20];
  printf("Content-type: text/html%c%c",10,10);

  /*
    This file is used to record how many users have given reports;
    the number of records is used as the name of the record file
  */

  if((recordNo=fopen("../htdocs/no_record","r+"))==NULL){
          printf("Open file error!\n");
     exit(1);
  }

  if(fscanf(recordNo,"%d",&no_record)<=0){
    no_record=1000;
    printf("Error");
  }

  sprintf(fileName, "../htdocs/body/record%d.html" ,no_record);
  sprintf(fileUrl, "../body/record%d.html" ,no_record);
  fseek(recordNo, 0L, SEEK_SET );
  fprintf(recordNo, "%d" ,++no_record);

  fclose(recordNo);


  /*
    This file store the title part of the page
  */
  title=fopen("../htdocs/title.html","r+");

  /*
```

```
  For every report, a new file is generated to store it
*/
body=fopen(fileName,"w+");

/*
  Add the new content before </body>and </html>
*/
fseek(title,-17,SEEK_END);

if(strcmp(getenv("REQUEST_METHOD"),"POST")) {
  printf("This script should be referenced with a METHOD of POST.\n");
  exit(1);
}



if(strcmp(getenv("CONTENT_TYPE"),"application/x-www-form-urlencoded")) {
  printf("This script can only be used to decode form results. \n");
  exit(1);
}
cl = atoi(getenv("CONTENT_LENGTH"));

for(x=0;cl & (!feof(stdin));x++) {
  m=x;
  entries[x].val = fmakeword(stdin,'&',&cl);
  plustospace(entries[x].val);
  unescape_url(entries[x].val);
  entries[x].name = makeword(entries[x].val,'=');
}


printf("<H1>Thanks, your report has been added!</H1>\n");
// printf("<A HREF=\"body.html\">See list of problems and solutions</A>");
// printf("<A HREF=\"465form.htm\">Input another problem</A>");
// printf("<A HREF=>Go to course webpage</A>");


fprintf(title,"\n<A HREF=%s> %s </A>\n<P>\n",fileUrl,entries[0].val);
fprintf(title,"</BODY>\n</HTML>");

/*
  Generate a new record file
*/
fprintf(body,"<HTML>\n");
fprintf(body,"<TITLE>\n");
```

```
  fprintf(body,"</TITLE>\n");
  fprintf(body,"<BODY>\n");
  fprintf(body,"<P><FONT COLOR=\"BLUE\" SIZE=4 >Brief description of problem:
</FONT><P>");
  fprintf(body,"<FONT COLOR=\"black\" SIZE=3>%s\n</FONT>",entries[0].val);

  x=1;
  fprintf(body,"<P><FONT COLOR=\"BLUE\" SIZE=3 >Details of problem:</FONT>\n
<P>");
  fprintf(body,"<FONT COLOR=\"black\" >%s</FONT>\n",entries[x++].val);

  fprintf(body,"<P><FONT COLOR=\"BLUE\" SIZE=4 >Severity of the
problem:</FONT>\n <P>");
  fprintf(body,"<FONT COLOR=\"Black\" SIZE=3 >%s</FONT>\n",entries[x++].val);
  fprintf(body,"<P><FONT COLOR=\"BLUE\" SIZE=4>What was the cause of the
problem?</FONT>\n <P>");
  fprintf(body,"<FONT COLOR=\"black\" SIZE=3>%s</FONT>\n",entries[x++].val);
  fprintf(body,"<P><FONT COLOR=\"BLUE\" SIZE=4>How was the problem
handled?</FONT>\n <P>");
  fprintf(body,"<FONT COLOR=\"black\" SIZE=3>%s</FONT>\n",entries[x++].val);
  fprintf(body,"<P><FONT COLOR=\"BLUE\" SIZE=4>Did the problem prevent
completion of the lab?</FONT>\n<P>");
  fprintf(body,"<FONT COLOR=\"black\" SIZE=3>%s</FONT>\n",entries[x++].val);
  fprintf(body,"<P><FONT COLOR=\"BLUE\" SIZE=4>Recommendations for other
students:</FONT>\n <P>");
  fprintf(body,"<FONT COLOR=\"black\" SIZE=3>%s</FONT>\n",entries[x++].val);

  fprintf(body,"<P>\n<P>");
  //fprintf(body,"\n<A HREF=../title.html>Back\n");
/*
  for(x=1; x <= m; x++){
   fprintf(body,"%s\n",entries[x].val);
  }
*/
  fprintf(body,"</BODY>\n");
  fprintf(body,"</HTML>\n");

  fclose(title);
  fclose(body);
  exit(0);

}

char *makeword(char *line, char stop) {
int x = 0,y;
```

```
char *word = (char *) malloc(sizeof(char) * (strlen(line) + 1));

for(x=0;((line[x]) && (line[x] != stop));x++)
  word[x] = line[x];

word[x] = '\0';
if(line[x]) ++x;
y=0;

while(line[y++] = line[x++]);
return word;
}

char *fmakeword(FILE *f, char stop, int *cl) {
int wsize;
char *word;
int ll;

wsize = 102400;
ll=0;
word = (char *) malloc(sizeof(char) * (wsize + 1));

while(1) {
  word[ll] = (char)fgetc(f);
  if(ll==wsize) {
    word[ll+1] = '\0';
    wsize+=102400;
    word = (char *)realloc(word,sizeof(char)*(wsize+1));
    }
  --(*cl);
  if((word[ll] == stop) || (feof(f)) || (!(*cl))) {
    if(word[ll] != stop) ll++;
    word[ll] = '\0';
    return word;
    }
  ++ll;
  }
}

char x2c(char *what) {
register char digit;

digit = (what[0] >= 'A' ? ((what[0] & 0xdf) - 'A')+10 : (what[0] - '0'));
digit *= 16;
digit += (what[1] >= 'A' ? ((what[1] & 0xdf) - 'A')+10 : (what[1] - '0'));
```

```
return(digit);
}

void unescape_url(char *url) {
register int x,y;

for(x=0,y=0;url[y];++x,++y) {
  if((url[x] = url[y]) == '%') {
    url[x] = x2c(&url[y+1]);
    y+=2;
    }
  }
url[x] = '\0';
}

void plustospace(char *str) {
register int x;
for(x=0;str[x];x++) if(str[x] == '+') str[x] = ' ';
}
```

6.6 CprE 489. Computer Networking

For the Computer Networking and Data Communications course, a communications protocol will be taught using the proposed approach. Ability to understand protocols is a core part of this type of course. Student learning of the protocol will be assessed using a homework assignment and a quiz. For all courses involved, instructor feedback will be obtained on both their satisfaction with the proposed approach and its utility. In the future, controlled experimentation would be desirable; this will be proposed to external funding agencies for follow on work.

Consider the problem of learning to understand a digital communication protocol. This appendix gives an outline of how software engineering concepts can be used as pedagogical tools to aid students in acquiring such an understanding.

The software life cycle begins with a rough account of what the product will do (often called the requirements analysis or functional requirements). Applied to the problem of teaching students a new protocol, this suggests beginning by reviewing the point of the protocol, in essence providing them with a requirements analysis. **Scenarios** and **rapid prototyping** might be good choices in illustrating the requirements. A rapid prototype approach might illustrate the protocol in slow motion, for example.

The next thing in a typical software engineering life cycle would be the detailed requirements, or **specifications**. Specifications tell what a piece of software will do but not how it will do it. Thus specifications in this case would flesh out the point of the protocol with all significant details. Once the students understand what the protocol does via the specs, they should be in a better position to understand how it does what it does. Depending on the problem, appropriate and widely used software engineering techniques useful in communicating the specifications include **data flow diagrams, entity-relationship modeling, finite state machines**, and **petri nets**. For the communications protocol problem, finite state machines might be a good choice due to their ability to describe the problem while remaining understandable to the students.

Describing how the protocol does what it does corresponds to the **design** phase of the software life cycle. The first subphase of design is **architectural design** in which the software system is broken down into **modules**. From the standpoint of a digital communication protocol, the modules might include one that runs on the sender and one on

the receiver. Students would be better equipped to understand the details of how the communication protocol works if they understand a modular breakdown of it first. A software engineering approach to doing this would likely use either **data flow diagrams**, or **Unified Modeling Language (UML) sequence diagrams** as a description language, given the problem of teaching a communication protocol. Other problems might call for **transaction analysis, abstract data types**, **class diagrams** with method descriptions, **collaboration graphs,** or some combination.

The second design subphase is the detailed design in which the details are given but not the actual program code. A **stepwise refinement** approach to presenting the detailed design might be helpful. This amounts to presenting something in successively more detail. This could be done with **flow charts** or in **Program Description Language** (**PDL**), often called **pseudocode**. PDL is basically a description that uses control statements of a given programming language, with other aspects of the code described in concise English rather than actual code. Flow charts might be good for visual students, while PDL might be more suitable for sequential learners or ones who learn well from printed text. Perhaps presenting a flow chart or PDL, and then converting it to the other form during the lecture, would be useful.

The reader may notice that while many of the concepts mentioned are specific to software engineering, some are more generic to engineering methodology. This is good, as it will facilitate the understanding and use of these concepts in the curriculum. The reader may also notice that it is not necessary to discuss details of the protocol itself as we describe how software engineering concepts could be brought to bear in teaching the protocol - it is enough just to know that it is a communication protocol that is under consideration. Should this be

the case for many important computing concepts, as expected, that will significantly facilitate the process of disseminating the strategy of software engineering throughout the curriculum: descriptions of the use of software engineering concepts in teaching other concepts will then be easily understandable to educators with specializations in a wide variety of specialties.

Arguably, certain aspects of this example could be derived from common sense pedagogy and generic engineering as much as to use of software engineering concepts specifically. Yet such "common sense" is easily overlooked. If explicitly applying software engineering concepts leads to better use of common sense engineering, this is yet another advantage of using software engineering across the curriculum.

7 Conclusion

The software development life cycle involves all the steps from client's requirement to the retirement of the software. Many other products have the similar life cycle, so the software life cycle model can be used in many fields to promote the efficiency and ease product development. At the same time many concepts in software engineering can be used to handle some specific problems. It is very useful to apply software engineering concepts in other courses. These techniques can be used to improve the understanding of some difficult problems in the course work; help the students on term projects and programming assignment; also students learn the way to solve problems using software engineering thinking.

8 Bibliography

1. Arnow, D.M., P. Whitlock, and C. Tretkoff, Using Modules to Integrate Distributed Computing in the Undergraduate CS Curriculum, First Annual Northeastern Small College Computing Conference (CCSCNE 96), April 19-20, 1996, University of Hartford, CT.

Consortium for Computing in Small Colleges.
http://www.sci.brooklyn.cuny.edu/~arnow/ED/CCSCNE96/NSCCC96.PAPER.html.

2. Bagert, D.J., T.B. Hilburn, G. Hislop, M. Lutz, M. McCracken, and S. Mengel, Guidelines for Software Engineering Education Version 1.0, Technical Report CMU/SEI-99-TR-032, Software Engineering Institute, Carnegie-Mellon University, 1999.
http://www.sei.cmu.edu/publications/documents/99.reports/99tr032/99tr032abstract.html

3. Bahrami, A., Object Oriented Systems Development, McGraw-Hill, 1999.

4. Barnes, B., G. Engel, M. Griss, R. LeBlanc, T. Wasserman, and L. Werth, Draft Software Engineering Accreditation Criteria, Computer 31 (4), May 1998, 73-77.

5. Berghel, H., "New Wave Prototyping: The Use and Abuse of Vacuous Prototypes," Interactions, 1:2 1994, pp. 49-54.
http://www.acm.org/~hlb/publications/new-wave/new-wave.html

6. Braxton, S. and D. Stone, ImpactCS, http://www.seas.gwu.edu/~impactcs/index.html, Web site.

7. Cushing, J., Seamless Integration of (Cleanroom) Software Engineering Techniques into a Computer Science Curriculum, Integrating Recent Research Results: An NSF-CISE Educational Infrastructure Workshop, July 7-11, 1997, Evergreen State College, Olympia, WA. http://www.evergreen.edu/user/CISE/cleanroom.html.

8. Grodzinsky, F. S., Project Teams: How to Build, Use, and Evaluate Them in Courses Across the Computer Science Curriculum, The Consortium for Computing in Small Colleges Third Annual Northeastern Conference (CCSCNE 98), April 24-25, 1998, Sacred Heart University, Fairfield, CT. Consortium for Computing in Small Colleges.
http://orion.ramapo.edu/~csconf/ccscne/98.

9. Horan, P., A. Goold, and Y. Yang, Integrating Software Engineering Throughout the Computing Curriculum, technical report TRC97/01, 1997, Deakin University, School of Computing and Mathematics, Victoria, Australia.

10. Huff, C. R., Martin, C.D. and Project ImpactCS Steering Committee. Computing consequences: A framework for teaching ethical computing (First Report of the Impact CS Steering Committee). *Communications of the ACM,* Vol.38, no.12, p.75-84, Dec., 1995.

11. Joint Task Force on Computing Curricula, Computing Curricula 2001, sponsored by IEEE Computer Society and by ACM, 2000. http://computer.org/education/cc2001/report/index.html.

12. Liu, P. and J.F. Peters, Risk Analysis in Ice-Melting HVDC Transmission Lines, Proc. CCECE'99, Edmonton, Alberta, May, 1999.

13. Martin, C. D., Huff, C. Gotterbarn, D., Miller, K. and Project ImpactCS Steering Committee, Implementing a tenth strand in the computer science curriculum (Second Report of the Impact CS Steering Committee). *Communications of the ACM,* Vol.39, no.12, p. 75-84, Dec., 1996.

14. McCauley, R. A., C. Archer, N. Dale, R. Mili, J. Roberge, and H. Taylor, The Integration of Software Engineering Principles Throughout the Undergraduate Computer Science Curriculum, *SIGCSE Bulletin* 27(1), 1995, 364-365.

15. Ourusoff, N., Organizer, 1 st Maine Higher Education Faculty/Student Development Workshop on "Teaching the O-O Paradigm Across the Computing Curriculum," May 1999. http://users.uma.maine.edu/faculty/nourusoff.

16. Pedrycz, W. and J.F. Peters (Eds.), *Computational Intelligence and Software Engineering*. Singapore: World Scientific Publishing Co. Pte. Ltd., 1998.

17. Peters, J.F., R. Agatep, S. Cormier, N. Dack, F. Kaikhosrawkani, N. Lao, O. Orenstein, P. Thang, V. Wan, and W.Y. Wong, Air Traffic Control Trainer Software Development: Concepts, blackboard architecture and Java prototype. Proc. of the Canadian Conf. on Electrical & Computer Engineering (CCECE'98), Waterloo, Ontario, 25-28 May 1998, 601-604.

18. Peters, J.F. and W. Pedrycz, Computational Intelligence. In: J.G. Webster (Ed.), *Encyclopedia of Electrica land Electronic Engineering.* 22 vols. NY: John Wiley & Sons, Inc., 1999, http://www.wiley.com/ee/engineering.htm.

19. Peters, J.F. and W. Pedrycz, *Software Engineering: An Engineering Approac*h. New York: John Wiley & Sons, Inc., 2000.

20. Peters, J.F. and S. Ramanna, Time-Constrained Software Cost Control System: Concepts and Roughly Fuzzy Petri Net Model. In: L.A. Zadeh, K. Hirota, E. Sanchez, P.-Z. Wang, and R.R. Yager (Eds.), Advances in Fuzzy Systems—Applications and Theory, Vol. 16. Singapore: World Scientific Publishing Co. Pte. Ltd., 1998, 339-369.

21. Peters, J.F., J. Wong, and S. Ramanna, Evolution of Competing Situated Robots: Concepts, and Experiments with a Java Applet. Proc. SMC'98, San Diego, California, October 1998, 3371-3374.

22. Placide, E., The Merging/Integration of Software Engineering Across the Computer Science Curriculum, ADMI 99 (The Symposium on Computing at Minority Institutions), June 3-6, Duluth, MN. http://cs.fdl.cc.mn.us/admi99/1999SCHEDULE121.html.

23. Schach, S., Classical and Object-Oriented Software Engineering with UML and Java, McGraw-Hill, 1999.

24. Strigel, W., What's the Problem: Labor Shortage or Industry Practices?, Guest Editor's Introduction, *IEEE Software* 16 (3) (May/June 1999) 52-54.

25. Thompson, S. and S. Hill, Functional Programming Through the Curriculum, in P. H. Hartel, and R. Plasmeijer, eds., *Functional Programming Languages in Education,* Lecture Notes in Computer Science number 1022, Springer-Verlag, 1995, 85-102. http://www.cs.ukc.ac.uk/pubs/1995/214/index.html.

26. Voas, J., Certification: Reducing the Hidden Costs of Poor Quality, Guest Editor's Introduction, *IEEE Software* 16 (4) 22-25, July/August 1999.

27. Werth, L.H., Integrating Software Engineering into Introductory Computer Science Courses, *Computer Science Education* 8 (1), March 1998.

REFERENCES

Berleant, Daniel, When Web Searches Return Hundreds of Documents: Extracting the Value, In preparation, 2001.

Berleant, Daniel & Jinping Huang, Ji Mu, Rajesh Potti, Xidan Zhou, and Zhong Gu, Multibrowsers: Reader Interaction with Documents via Direct Multidisplay. In preparation, 2001.

Berleant, Daniel, Dialogue With Documents: Customized Interaction With Texts, In preparation, 2001.

Cavnar, W., Using An N-Gram-Based Document Representation With A Vector Processing Retrieval Model, in NIST Special Publication 500-226: Overview of the Third Text Retrieval Conference (TREC-3), 1994, 269-278. http://trec.nist.gov/pubs/trec3/t3_proceedings.html.

Charoenkitkarn, N., M. Chignell, and G. Golovchinsky, Interactive Exploration as a Formal Text Retrieval Method: How Well can Interactivity Compensate for Unsophisticated Retrieval Algorithms, in NIST Special Publication 500-226: Overview of the Third Text Retrieval Conference (TREC-3), 1994, 179-199. http://trec.nist.gov/pubs/trec3/t3_proceedings.html.

Cohen, J. D., Highlights: Language- and Domain-Independent Automatic Indexing Terms for Abstracting. Journal of the American Society for Information Science 46(3), 1995, 162-174.

Crowder, G. and C. Nicholas, Resource Selection in CAFÉ: An Architecture for Network Information Retrieval, Proceedings of the Network Information Retrieval Workshop, SIGIR 96, August 1996.

Cubiceye, 2ce™, April 2001.

http://www.2ce.com.

D'Amore, R. and Mah, C., One-time complete indexing of text: Theory and practice, in Proceedings of SIGIR 1985, 155-164.

Damashek, M., Gauging similarity with n-grams: Language-independent categorization of text, Science, 267 (1995), pp. 843 – 848.

Ebert, D.S., C. D. Shaw, A. Zwa, E.L. Miller, and D.A. Roberts, Interactive Volumetric Information Visualization for Document Corpus Management, Proceedings of Graphics Interface 97, Kelowna, B.C., May 1997, 121-128.

http://www.dgp.toronto.edu/gi/gi97/proceedings/.

Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach And T. Berners-Lee. "Hypertext Transfer Protocol -- Http/1.1." Internet-Draft. 2000.

http://www.w3.org/protocols/http/1.1/draft-ietf-http-v11-spec-rev-06.txt.

Fox, K., O. Rieder, M. Knepper, and E. nowberg "SENTINEL: A Multiple Engine Information Retrieval and Visualization System," Journal of the American Society of Information Science, 50(7), May 1999, 616-625.

Grossman, D.A., A Parallel DBMS Approach to IR in TREC-3, in NIST Special Publication 500-226: Overview of the Third Text Retrieval Conference (TREC-3), 1994, 279-288.

http://trec.nist.gov/pubs/trec3/t3_proceedings.html.

Grossman, D.A., D.O. Homes, O. Frieder, M.D. Nguyen, and C.E. Kingsbury, Improving Accuracy and Run-Time Performance for TREC-4, in NIST Special Publication 500-236: The Fourth Text Retrieval Conference (TREC-4), 1995, 433-448.

http://trec.nist.gov/pubs/trec4/t4_proceedings.html.

Huffman, S., Acquaintance: Language-Independent Document Categorization by N-Grams, in NIST Special Publication 500-236: The Fourth Text Retrieval Conference (TREC-4), 1995, 359-372. http://trec.nist.gov/pubs/trec4/t4_proceedings.html.

Huffman, S. and M. Damashek, Acquaintance: A novel vector-space n-gram technique for document categorization, in NIST Special Publication 500-226: Overview of the Third Text Retrieval Conference (TREC-3), 1994, 305-310. http://trec.nist.gov/pubs/trec3/t3_proceedings.html.

Kandogan, Eser & Shneiderman, Ben, Elastic Windows: Evaluation of Multi-Window Operations, In Proceedings of CHI'97 (Atlanta, March 22-27 1997), pages 250-257.

Labrou, Y. and T. Finin, Experiments on using Yahoo! categories to describe documents, in IJCAI-99 Workshop on Intelligent Information Extraction, July 1999. http://www.aifb.uni-karlsruhe.de/WBS/dfe/iii99/.

Mayfield, J., P. McNamee, Indexing Using Both N-Grams and Words, in NIST Special Publication 500-242: The Seventh Text Retrieval Conference (TREC 7), 1998, 419-424. http://trec.nist.gov/pubs/trec7/t7_proceedings.html.

Mukherjea, Sougata and Yoshinori Hara, Visualizing World-Wide Web Search Engine Results, Information Visualization, 1999. Proceedings. 1999 IEEE International Conference on, 1999.

Ng, K.B. and P.B. Kantor, Two Experiments on Retrieval With Corrupted Data and Clean Queries in the TREC-4 Adhoc Task Environment: Data Fusion and Pattern Scanning, in NIST Special Publication 500-236: The Fourth Text Retrieval Conference (TREC-4), 1995, 499-508.

http://trec.nist.gov/pubs/trec4/t4_proceedings.html.

Pearce, C. and E. Miller. The telltale dynamic hypertext environment: Approaches to scalability. In James Mayfield and Charles Nicholas, editors, Advances in Intelligent Hypertext, Lecture Notes in Computer Science. Springer-Verlag, 1997. 6.

Pearce, C. and C. Nicholas, TELLTALE: Experiments in a Dynamic Hypertext Environment for Degraded and Multilingual Data, Journal of the American Society for Information Science (JASIS), April 1996. 263-275.

Sibun, P., & Reynar, J., Language identification: Examining the issues. In Symposium on Document Analysis and Information Retrieval, 1996, 125-135.

Salton, G., Automatic Text Processing, Addison Wesley Pub. Co., 1989.

Suen, C.Y., N-gram statistics for natural language understanding and text processing, IEEE Trans. on Pattern Analysis and Machine Intelligence, PAMI 1 (2) 1979, 164-172.

Susan Maze, David Moxley, And Donna J. Smith, Authoritative Guide To Web Search Engines, Neal-Schuman Publishers Inc., 1997.

Zamir, Oren & Etzioni, Oren, Grouper: A Dynamic Clustering Interface to Web Search Results, WWW8 / Computer Networks 31(11-16), 1999, 1361--1374.

Zamora, E.M., J.J. Pollock, and A. Zamora, The use of trigram analysis for spelling error detection, Information Processing and Management 17 (6) 1981, 305-316.

ACKNOWLEDGEMENT

I want to express my most sincere gratitude to my advisor, Dr. Daniel Berleant. From Dr. Berleant, I learned not only the specific knowledge about how to work on specific projects, but also the way to analyze and solve problems in the engineering field. He is always kind to me, treats me like a peer worker. All the materials I learned from him will benefit to me for my whole life. I also want to thank Dr. Steve Russell who gave me many very useful hints and advice, which help me a lot on software engineering project.