ABSTRACT

APPLICATIONS OF INTERNET TECHNOLOGY FOR REQUIREMENTS ELICITATION

by
Deepak Pandit

During the Requirements Elicitation part of a project various stakeholders need to be able to communicate their requirements to the developers, and the developers need to be able communicate their understanding back to the stakeholders. Communication between the various members of the project is the key factor during the Requirements Elicitation part of a project. Easing communications between stakeholders and developers makes the process of eliciting requirement easier, leading to better requirements specification and eventually a better product.

The Requirements Elicitation Process through Internet (REPI) web site has been designed and implemented to explore this idea. The prototype version of REPI guides project members through the elicitation phase using the Software Engineering Institute's framework for Requirements Elicitation. The REPI web site forces stakeholders to explicitly describe the requirements and encourage early discussion between stakeholders and developers. This decreases the likelihood of misunderstood requirements, leading to better requirements specification.

APPLICATIONS OF INTERNET TECHNOLOGY FOR REQUIREMENTS ELICITATION

by
Deepak Pandit

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer and Information Science

Department of Computer and Information Science

January 1998

APPROVAL PAGE

APPLICATIONS OF INTERNET TECHNOLOGY FOR REQUIREMENTS ELICITATION

Deepak Pandit

Dr. Murat M. Tanik, Thesis Advisor — Date
Department of Computer and Information Science
New Jersey Institute of Technology

Dr. Franz Kurfess, Committee Member — Date
Department of Computer and Information Science
New Jersey Institute of Technology

Dr. Donald H. Sebastian, Committee Member — Date
Department of Industrial and Manufacturing Engineering
New Jersey Institute of Technology

BIOGRAPHICAL SKETCH

Author: Deepak Pandit

Degree: Master of Science

Date: January 1998

Undergraduate and Graduate Education:

- Master of Science in Computer and Information Science, New Jersey Institute of Technology, Newark, NJ, 1998
- Bachelor of Science in Computer and Information Science, New Jersey Institute of Technology, Newark, NJ, 1996

Major: Computer and Information Science

This thesis is dedicated to my family

ACKNOWLEDGMENT

The author wishes to express his sincere gratitude to his advisor, Dr. Tanik, for all his guidance and support. The author wishes to thank him for all his suggestions and help during the thesis writing process.

The author appreciates all the suggestions and help from Dr. Kurfess and would like thank him for serving as a member of the committee. The author wishes to thank Dr. Sebastian for serving as a committee member and for his suggestions. Special thanks to Professor Jololian and Mr. Smith for their participation during my defense.

TABLE OF CONTENTS

Chapter	Page
1 INTERNET TECHNOLOGIES	1
1.1 Web Protocols and Standards	3
1.1.1 HyperText Transport Protocol	4
1.1.2 Cookies	9
1.1.3 Common Gateway Interface	11
1.1.4 HyperText Markup Language	14
1.2 Messaging Protocols and Standards	35
1.2.1 Simple Mail Transfer Protocol	37
1.2.2 Post Office Protocol	41
1.2.3 Internet Message Access Protocol	44
1.2.4 POP3 vs. IMAP4	47
1.2.5 Internet Media Type	50
1.3 Languages	57
1.3.1 Java	58
1.3.2 JavaScript	70
2 REQUIREMENTS ELICITATION	76
2.1 Requirements Elicitation Framework	78
2.2 Requirements Elicitation Process Model	80
2.3 Requirements Elicitation Methodology and its Methods	83

TABLE OF CONTENTS
(Continued)

Chapter		Page
	2.3.1 Fact Finding	84
	2.3.2 Gathering and Classification	86
	2.3.3 Evaluation and Rationalization	88
	2.3.4 Prioritization and Planning	89
	2.3.5 Integration and Validation	89
3	REQUIREMENTS ELICITATION WITH INTERNET TECHNOLOGIES	91
	3.1 Introduction	91
	3.2 REPI Web Site Description	92
	3.2.1 Login Screen	98
	3.2.2 Menu Screens	98
	3.2.3 Fact Finding Phase	100
	3.2.4 Gathering and Classification Phase	107
	3.2.5 Evaluation and Rationalization Phase	117
	3.2.6 Prioritization and Planning Phase	121
	3.2.7 Integration and Validation Phase	125
	3.2.8 Information Pages of the REPI Web Site	129
	3.2.9 Other Components of the REPI Web Site	133
	3.3 REPI Web Site Evaluation	143
	3.3.1 Web Site Design	143

TABLE OF CONTENTS
(Continued)

Chapter			Page
4	CONCLUSION AND FUTURE WORK		152
	4.1	Benefits of the REPI Web Site	152
	4.2	Limitations of the REPI Web Site	155
	4.3	Future Work	158
APPENDICES	A.1	CGI Example	160
	B.1	HTML Lists Example	160
	B.2	HTML Table Example	161
	B.3	Netscape Frame Example	162
	B.4	HTML Form Example	165
	C.1	ESMTP Example	166
	D.1	IMAP4 States Example	167
	D.2	IMAP4rev1 Example	167
	E.1	Java Inheritance Example	168
	E.2	Java Thread Example	170
	E.3	View of the Java Environment	172
	F.1	"Login Screen" of the REPI Web Site	174
	G.1	Menu Pages of the REPI Web Site	179
	H.1	User's Fact Finding Pages	186
	H.2	User's Gathering and Classification Pages	191

TABLE OF CONTENTS
(Continued)

Chapter		Page
H.3	User's Evaluation and Rationalization Pages	202
H.4	User's Prioritization and Planning Pages	203
H.5	User's Integration and Validation Pages	204
I.1	Developer's Fact Finding Pages	205
I.2	Developer's Gathering and Classification Pages	206
I.3	Developer's Evaluation and Rationalization Pages	207
I.4	Developer's Prioritization and Planning Pages	208
I.5	Developer's Integration and Validation Pages	213
J.1	REPI Web Site's Information Pages	214
K.1	Read and Send Messages Pages	221
K.2	What's New Page	223
K.3	Todo Tasks Pages	224
K.4	Help Pages	225
K.5	Logout and Error Message Pages	226
L.1	REPI Web Site's Style Sheet and JavaScript Source Code	227
REFERENCES		234

LIST OF TABLES

Table		Page
1	JavaScript Objects	71
2	Comparison of JavaScript and Java	75
3	Requirements Elicitation Process Model's Tasks	82
4	SEI Compared with REPI for the Fact Finding Phase	101
5	SEI Compared with REPI for the Gathering and Classification Phase	108
6	SEI Compared with REPI for the Evaluation and Rationalization Phase	117
7	SEI Compared with REPI for the Prioritization and Planning Phase	121
8	SEI Compared with REPI for the Integration and Validation Phase	126

LIST OF FIGURES

Figure		Page
1	View of the Web	3
2	HTTP Dialogue Example	6
3	HTTP Event Flow Example	7
4	Cookie Example	11
5	HTML Example	15
6	HTML Imagemap Example	24
7	CSS Example Part 1 of 2	27
8	CSS Example Part 2 of 2	28
9	JSSS Example	29
10	CSS-P Example	31
11	JASS Example	31
12	Dynamic HTML Example	34
13	SMTP Model	38
14	SMTP Example	39
15	Email Header Example	40
16	POP3 Dialogue Example	42
17	POP3 Session State Diagram	44
18	Internet Media Type Example 1	52
19	Internet Media Type Example 2	56
20	Java Application Example	66

LIST OF FIGURES
(Continued)

Figure		Page
21	Java Applet Example Part 1 of 2	69
22	Java Applet Example Part 2 of 2	69
23	JavaScript Example Part 1 of 2	73
24	JavaScript Example Part 2 of 2	74
25	Requirements Elicitation Framework	79
26	Requirements Elicitation Process Model	81
27	REPI Web Site Structure Overview	93
28	REPI Web Site Client Side Structure	95
29	REPI Web Site Developer Side Structure	95
30	"Login Screen" of the REPI Web Site	97
31	REPI Web Site's Menu Screens	99
32	Task 5, "Identify Similar Systems," of the Fact Finding Phase	104
33	Task 1, "Requirements List," of the Gathering and Classification Phase	109
34	Task 2, "Add Requirement," of the Gathering and Classification Phase	112
35	Task 2, "Capture Rationale," of the Evaluation and Rationalization Phase	119
36	Task 1, "Prioritize Requirements," of the Prioritization and Planning Phase	123
37	Task 2, "Plan development stages," of the Prioritization and Planning Phase	125
38	"Requirements Information" Page of the REPI Web Site	131
39	"Send Messages" Page of the REPI Web Site	135

LIST OF FIGURES
(Continued)

Figure		Page
40	"Project Members" Page of the REPI Web Site	140
41	CGI Example	160
42	HTML Lists Example	161
43	HTML Table Example	162
44	Netscape Frame Example Part 1 of 3	163
45	Netscape Frame Example Part 2 of 3	164
46	Netscape Frame Example Part 3 of 3	164
47	HTML Form Example	166
48	ESMTP Example	166
49	IMAP4 States Example	167
50	IMAP4rev1 Example	168
51	Java Inheritance Example	169
52	Java Thread Example	171
53	View of the Java Environment, Part 1 of 2	172
54	View of the Java Environment, Part 2 of 2	173
55	"Login Screen" on Various Platforms Using Different Browsers	174
56	"Login Screen" of the REPI Web Site	175
57	HTML Source Code for "Login Screen"	178
58	User's Main Menu Screen and Developer's Main Menu Screen	179

LIST OF FIGURES
(Continued)

Figure		Page
59	HTML Source Code for "User's Main Menu"	180
60	HTML Source Code for "User's Main Menu" Left Frame	182
61	HTML Source Code for "User's Main Menu" Title Frame	183
62	HTML Source Code for "User's Main Menu" Right Frame	185
63	User's Fact Finding Menu Screen	186
64	HTML Source Code for "User's Fact Finding Menu" Right Frame	189
65	Five Tasks of the User's Fact Finding Phase	190
66	Task 1 of the User's Gathering and Classification Phase	191
67	HTML Source Code for Task 1 of User's Gathering and Classification Phase	197
68	Task 2 of the User's Gathering and Classification Phase	197
69	HTML Source Code for Task 2 of User's Gathering and Classification Phase	202
70	Two Tasks of the User's Evaluation and Rationalization Phase	202
71	Task 1 of the User's Prioritization and Planning Phase	203
72	Three Tasks of the User's Integration and Validation Phase	204
73	Four Tasks of the Developer's Fact Finding Phase	205
74	Task 1 of the Developer's Gathering and Classification Phase	206
75	Three Tasks of the Developer's Evaluation and Rationalization Phase	207
76	Task 1 of the Developer's Prioritization and Planning Phase	208
77	Task 2 of the Developer's Prioritization and Planning Phase	209

LIST OF FIGURES
(Continued)

Figure		Page
78	HTML Source Code for Task 2 of Developer's Prioritization and Planning Phase	212
79	Task 3 of the Developer's Prioritization and Planning Phase	212
80	Task 1 of the Developer's Integration and Validation Phase	213
81	"Requirements Information" Page of the REPI Web Site Part 1 of 2	214
82	"Requirements Information" Page of the REPI Web Site Part 2 of 2	225
83	HTML Source Code for the "Requirements Information" Page	219
84	"Category Information" Page of the REPI Web Site	219
85	"User Information" Page of the REPI Web Site	220
86	"Read Messages" Page of the REPI Web Site	221
87	"Send Messages" Page of the REPI Web Site	222
88	"What's New" Page of the REPI Web Site	223
89	Five TODO Tasks of the REPI Web Site	224
90	Help Pages for the REPI Web Site	225
91	Error Message for the REPI Web Site	226
92	"Logout Screen" of the REPI Web Site	227
93	Style Sheet for the REPI Web Site	228
94	JavaScript Source Code for the REPI Web Site	233

CHAPTER 1

INTERNET TECHNOLOGIES

The Internet is a global network of networks connecting very large number of users worldwide using a simple standard common addressing system and communications protocol. Many networks are part of the Internet, including federal networks, regional networks, educational networks and some foreign networks [RICHMOND 97]. The Federal Networking Council (FNC), in consultation with members of the Internet and intellectual property rights communities, provides this definition:

> "Internet" refers to the global information system that -- (i) is logically linked together by a globally unique address space based on the Internet Protocol (IP) or its subsequent extensions/follow-ons; (ii) is able to support communications using the Transmission Control Protocol/Internet Protocol (TCP/IP) suite or its subsequent extensions/follow-ons, and/or other IP-compatible protocols; and (iii) provides, uses or makes accessible, either publicly or privately, high level services layered on the communications and related infrastructure described herein [LEINER 97].

Some of the "founding fathers" of the Internet have written a brief history of the Internet where they say "the Internet is at once a world-wide broadcasting capability, a mechanism for information dissemination, and a medium for collaboration and interaction between individuals and their computers without regard for geographic location" [LEINER 97].

Intranet can be thought of as a local area network based on Internet technology. It uses the same technologies used in the Internet; but its servers are limited to connections inside a company's networks; in effect, a private Internet for a company. Intranets are

the replacement for a company's Local Area Network (LAN). Intranets provide the same functionality as a LAN, but they are easier and cheaper. The use of open Internet standards provides Intranet users with more choices, easier setup and maintenance, lower cost of application deployment and management, cross platform access to information and applications, easier access to information, and lower training costs [ORACLE 96]. Extranets are Intranets, from different companies, joined together using the Internet, for the purpose of better integration between close business partners. The underlying technologies of all these different types of networks are the same, at least at the application level. This thesis views, Internets, Intranets and Extranets as being the same for the purposes of the REPI web site implementation.

With the growth of the Internet, the most likely question is "When will the Internet do this . . .?" and not "Can the Internet do this . . .?" Companies and people are using the Internet for many things, including uploading and downloading of newly developed software, gathering ideas and specifications for new software, or beta testing using selected clients or the general public. The Internet is many things to many people. In this thesis, the Internet is viewed as a large-scale, open, multi-vendor environment for distributed application development. The Internet can be used for all types of communications needs, between developers, clients and end-users, during the application development process.

This chapter of the thesis describes the technologies involved in the Internet. Section 1.1 describes the World Wide Web's protocols and standards. Section 1.2

describes messaging protocols and standards used for email communications. Section 1.3 describes the programming languages used for Internet application development.

1.1 Web Protocols and Standards

The World Wide Web, also referred to as just the "web," is a huge collection of hypertext documents linked together. It is a completely distributed network of individual web sites and web pages, without any overall organization. Web sites are used by many people and organizations for many purposes. Large and small companies use their web sites as a virtual public relations office or as a virtual product showcase. Individual people and other organizations use web pages to publish information about themselves and their interests. Organizations are using the Internet as virtual offices and as a new platform of business applications that needs to be accessed from variety of locations and/or platforms.

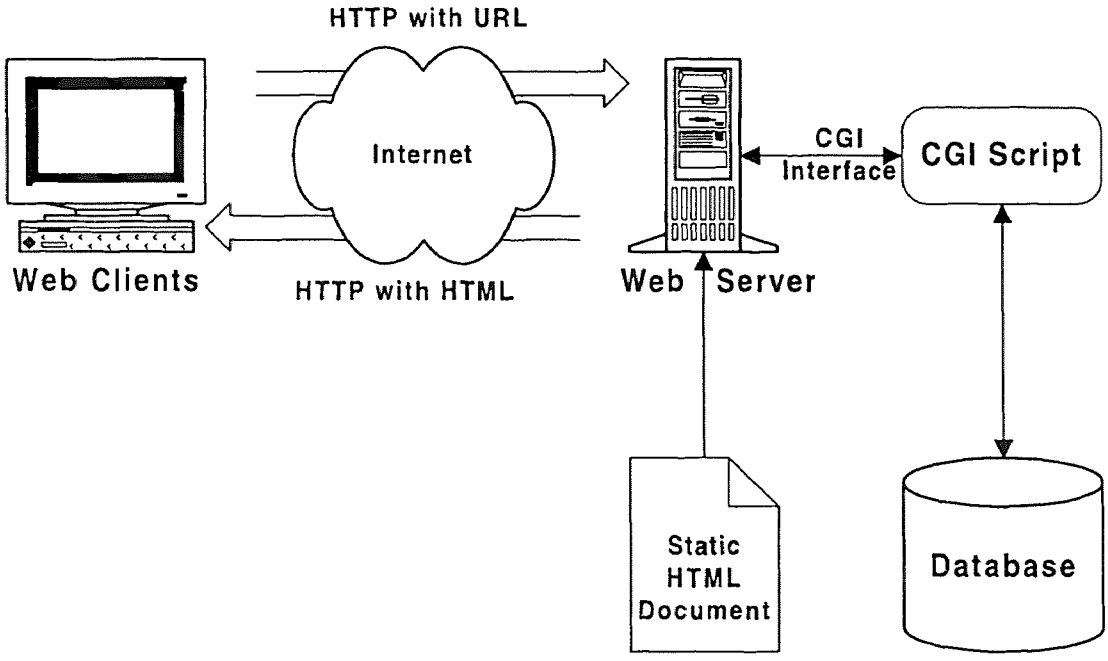


Figure 1: View of the Web. Source: Butterworth, Paul. "Web Access to the Core Business Infrastructure." Report MCS-0260-1. Forté Software Inc. September 1996.

Figure 1 shows an overview of the World Wide Web's technology. Web client software uses the HyperText Transport Protocol (HTTP) to transmit a request to the web server across the Internet. The web server software, also using HTTP, either returns a static HyperText Markup Language (HTML) document from its local disk or uses Common Gateway Interface (CGI) scripts to communicate with external applications such as database servers. This part of the chapter describes the protocols and standards of the World Wide Web. Section 1.1.1 describes the HyperText Transport Protocol. Section 1.1.2 describes Cookies, used for persistence state on the client side. Section 1.1.3 describes the Common Gateway Interface. Section 1.1.4 describes the HyperText Markup Language.

1.1.1 HyperText Transport Protocol

HyperText Transport Protocol (HTTP) is a standard method for requesting hypertext documents on the web and responding to such requests between two computers on the Internet. The protocol uses a two-tier architecture with a server application, called the web server, sending the hypertext documents requested by the client application, generally called the web browser. This protocol is a stateless protocol, meaning that the server does not maintain any information about the client. This makes keeping track of user behavior on the web difficult [RICHMOND 97]. HTTP is also a connectionless protocol. The implication of this is that for each document request, the client has to reconnect with the server. Since most of the web pages are composed of multiple items, such as image files, multiple connections are needed to retrieve a single web page. This

introduces more traffic between the client and the server; but this arrangement reduces the resource requirements at the server end. The advantage of this stateless and connectionless protocol is that the server can provide service to more clients with fewer connections because the server does not have to maintain idle connections to the client while the person is reading the contents of the web pages.

HTTP uses Internet Media Types as an open and extensible data typing mechanism and it is also used for type negotiations. This feature of HTTP makes it possible for it to be used as a generic document transportation protocol, not limited to hypertext documents such as HTML web pages. This freedom to transmit data of any type is one of the most significant advantages of HTTP [RICHMOND 97].

1.1.1.1 Current Version 1.0: As with any other protocol, this protocol has message formats and valid commands and requests. The usual sequence of communication is for the client to request a document using a Uniform Resource Locator (URL). The server responds with the requested document or with an error message. After this exchange, the connection between the client and the server is broken. Figure 1 shows an overview of the communications involved in a web dialogue.

The general format of a URL is:

scheme://host.domain:port/path/filename#anchor [NCSA 97].

Where *scheme* represents the type of resource being accessed. The web pages are delivered by the HTTP server; but other resources such as local files, gopher, Wide Area Information Service (WAIS), File Transfer Protocol (FTP), news, and telnet can also be

accessed using a URL. The *host.domain* refers to the Internet Protocol (IP) address of the server for the resource scheme. The *port* refers to the communication port used by the server; each type of service generally has a default communication port. The */path/filename* is a reference to the specific file being requested, using the standard UNIX style pathname conventions. The *anchor* section of the URL is used for web pages to refer to named sections of a long document. All parts of the URL, expect the *host.domain* part, are optional in today's web browsers.

```
GET http://megahertz.njit.edu HTTP/1.0

HTTP/1.0 200 Document follows
MIME-Version: 1.0
Server: CERN/3.0
Date: Monday, 21-Jul-97 11:26:48 GMT
Content-Type: text/html
Content-Length: 51039
Last-Modified: Monday, 21-Jul-97 03:02:03 GMT

<html>
<head>
<title>WWW HOME PAGE LISTING FOR megahertz.njit.edu</title>
</head>
<body>
  •
  •
  •
```

Figure 2: HTTP Dialogue Example

An example of a dialogue between the client and the server is shown in Figure 2. The HTTP Request header consists of the request method, URL for the target file, and the protocol and version information. The example, in Figure 2, shows a *GET* method for the URL, using the HTTP version 1.0 protocol. As part of the response header, the server returns the HTTP version, a status code and status message. The status codes can be divided into five categories with broad meaning: Information status codes, Success status

codes, Redirection status codes, Client error status codes, and Server error status codes [HERRMANN 96]. The example, in Figure 2, shows a status code, 200, with the status message "Document follows." The status code 200 is a general success code used for positive acknowledgments; the usual status message for this code is "OK." The status message is followed by the current date and the HTTP server name and version information. The next two headers identify the type and size of the data being sent by the server, followed by the last modified date of that file. The actual contents of the file follow the header information. Figure 3 shows the event flows in a client/server connection.

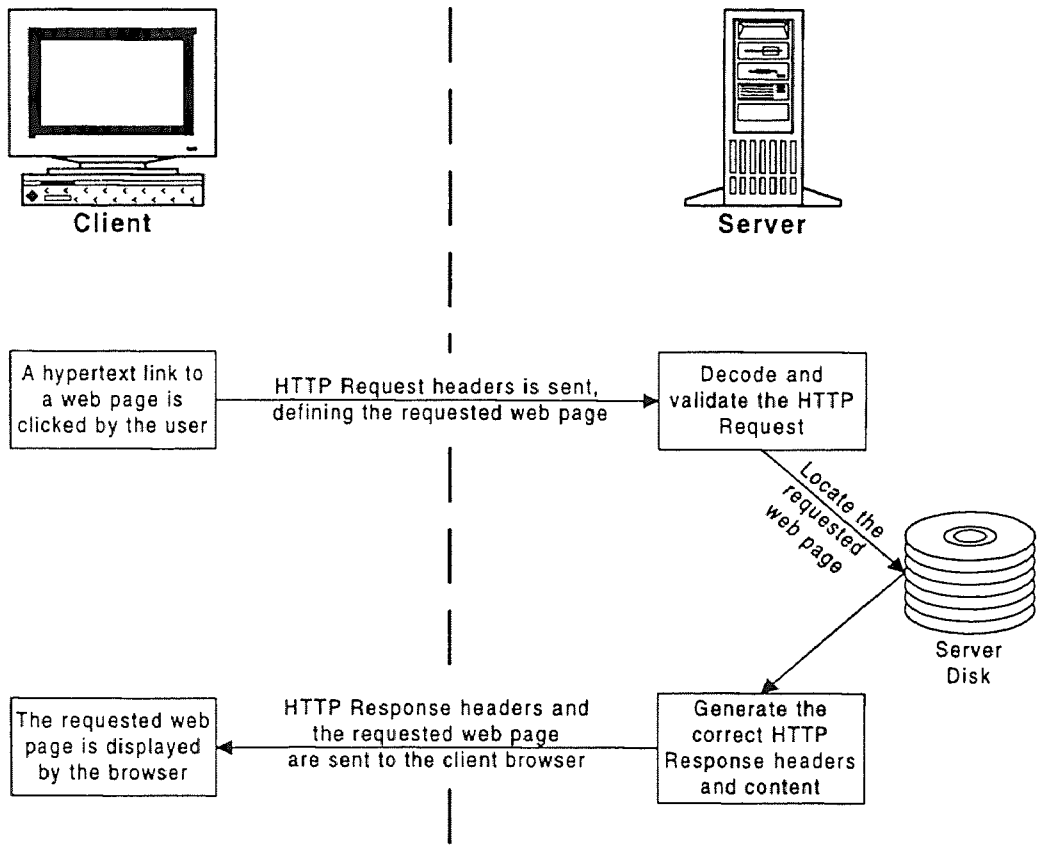


Figure 3: HTTP Event Flow Example. Source: Herrmann, Eric. *Teach Yourself CGI Programming with Perl in a Week.* Sams.net, Indianapolis, Indiana. 1996.

1.1.1.2 Next Version 1.1: Internet Engineering Task Force (IETF), working with World Wide Web Consortium (W3C), has created HTTP version 1.1, the newest version, as a proposed standard, documented in Request for Comment (RFC) 2068. The current version of the HTTP is no longer able to meet the demands of web users because of the continuous growth experienced recently.

HTTP transactions are the biggest consumers of the available Internet bandwidth. Measurements have shown that the web traffic on the Internet affects all layers, from low level transport protocols to high level application protocols. It has also been observed that HTTP transactions have high computational overhead in parsing HTTP messages [NIELSEN 97]. The purpose of the new version is to fulfill the demand and at the same time preserve the simplistic design of the original version. The general improvements made in HTTP/1.1 are the results of re-factoring the elements of HTTP/1.0 into separate layers and modules to produce a cleaner design that can be implemented more easily, yet remain flexible. HTTP is a stateless protocol that requires a reconnection for each and every web page that is downloaded. This constant connection/disconnection causes a big overhead for the TCP layer of the Internet, which was designed for connections lasting more than few seconds (as is usually the case with HTTP request/response cycle). The next generation HTTP (HTTP-NG) divides up the connection between client and server into many different channels and multiple requests can be sent and received using one connection [RICHMOND 97]. Persistent connections and pipelining are thought to be solutions to the problems of overhead. But persistent connections and pipelining will not solve all the problems; ". . . the reason is that HTTP/1.1 is designed to limit TCP

overhead produced by HTTP/1.0 but not protocol overhead due to HTTP itself" [NIELSEN 97]. The protocol overhead of HTTP itself is caused by verbose messages that are in human readable form instead of messages that are more efficient for the server to parse and process. The human readable format requires complicated parsers, thus increasing the CPU overhead for the protocol.

Many limitations still exist in the new version, because of an explicit design decision made to keep version 1.1 backward compatible with version 1.0. The caching model used in HTTP/1.1 has become very complex caused by maintaining backward compatibility with HTTP/1.0. The Protocol Extension Protocol (PEP) is one of the areas being worked on to create HTTP-NG. HTTP-NG bases its architecture on a distributed, object-oriented model, attempting to "break down the problem using layering and modularization simplifying the solution and reflecting current and near future usage of the web" [NIELSEN 97]. The PEP allows HTTP to be dynamically extended in such a way that applications that use PEP do not need agreement with the rest of the Internet. This allows extended and regular protocols to co-exist on the Internet. It allows applications to use the extensions without prior agreement; if an application does not support the extension the transaction can either be aborted or a minimum set of capabilities can be negotiated [NIELSEN 97].

1.1.2 Cookies

Cookies are a method of providing state information in the stateless protocol of the web. Cookies allow the server to transmit information to the client. The client re-transmits the

saved cookies when the server requests them. Another way of looking at cookies is to think of them as environment variable that can be set by the server on the client machine [HERRMANN 96]. On the client side the cookie is given a name and a value, which is stored in a file by the client browser. On the server side this name/value pair is set to the *HTTP-COOKIE* environment variable after receiving the information in a HTTP request header. This provides the server with a method to keep track of users connecting to the server. "The addition of a simple, persistent, client-side state significantly extends the capabilities of web-based client/server applications" [NETSCAPE 97].

The web server sets cookies by sending the *Set-Cookie* header as part of the HTTP response header, in reply to a web page requested by the browser. The *Set-Cookie* header has a name identifying the cookie and a value for that cookie. It has an expiration date, which defines the valid life time for this cookie; but the browser or the user could delete the cookie before that date. As part of the header, path and domain information is also sent. This information restricts the web sites and web pages that can retrieve this cookie. It also has an optional secure attribute which prevents it from being transmitted on insecure connections. The web browser sends back cookie information as part of the HTTP response header using the *Cookie* header. The *Cookie* header contains a series of name-value pairs, listing all cookies that match the path and domain information for the current web server and web page path. Figure 4 shows the syntax for both the *Set-Cookie* header and the *Cookie* header.

```
Set-Cookie: NAME=VALUE; expires=DATE
path=PATH; domain=DOMAIN_NAME; secure

Cookie: NAME1=OPAQUE_STRING1; NAME2=OPAQUE_STRING2
```

Figure 4: Cookie Example. Source: Persistent Client State: HTTP Cookies. http://home.netscape.com. Netscape Communications Corporation. 1997.

Cookies have many limitations in them and these limitations are of two types. The first type of limitations is the restrictions imposed by the standard. These restrictions are: only 20 cookies are allowed per server; only 300 cookies are allowed per client; and each cookie has to be less than 4 Kb. [NETSCAPE 97]. The second type of limitations is the use of the cookies. One problem is that cookies can not be forced down the user's throats. That is, users can prevent cookies being saved in their browsers and they can be easily modified. The second problem is that cookies are difficult to use to store complex information. Any information that needs to be maintained has to be translated into a set of character strings.

But even with the above limitations, cookies have been used in many ways and for many purposes. Some of its current uses are: to keep track of login and registration information, to maintain user preferences for a given web site, to keep track of previous visits. Some of the newer uses for cookies are in the area of on-line shopping. Cookies are used to maintain the contents of a user's "shopping bag."

1.1.3 Common Gateway Interface

One of the major limitations of regular HTML is that it does not allow users to interact with the web server. The Common Gateway Interface (CGI) is the standard interface

between the web server and external applications running on that server or other servers. Or as Herrmann puts it, "CGI programming is writing applications that act as interface or gateway programs between the client browser, web server, and a traditional programming application" [HERRMANN 96]. CGI provides a method for creating dynamic web pages based on input provided by the user and output provided by external applications [CGI].

The web browser gathers the user's input using HTML forms and passes this information to the web server using HTTP. The web server runs a CGI script to process the user's input and sends the results back to the web browser. This allows the possibility of interaction between users and the web site.

CGI provides the Server Side Includes (SSI) feature that enables web pages to contain dynamic content instead of the static content limitations of simply retrieving a pre-formatted HTML file from the server. Server Side Includes are special HTML-like commands that are executed by the server as it parses the HTML file before sending it as part of the HTTP response [HERRMANN 96]. The output from SSI commands replaces the SSI command call text in the HTML file; in effect the results of the commands are included as part of the web page. There are five types of SSI commands: *config*, *echo*, *exec*, *fsize*, *flastmod*, and *include*. Of these the *exec* command and the *include* command are the most powerful. The *exec* command allows the CGI programmer to execute any program on the server and merge its output into the web page. The *include* command allows the inclusion of other HTML files providing the web designer with the ability to use standard web page design elements easily. For example, a standard signature and copyright element could easily be attached to all the web pages of a site by using the

include command at the end of all the pages. Without this feature the designer would be forced to maintain these standard elements in each of the pages instead of maintaining it in one standard template. Appendix A.1 shows an example of some of the SSI commands available in CGI. In this figure the top part shows the main file which uses the *config*, *echo*, *fsize*, *flastmod*, and the *include* commands. The bottom part shows the target for the *include* command, a file that has the copyright information and a standard signature which uses the *exec* command to display a graphic signature.

One of the major problems with CGI is that of security [HERRMANN 96]. CGI has to capture information from the user and send this information to the server for processing. CGI exposes the captured information in the URL or in the hidden fields inside an HTML form. The information that is sent to the script can be easily edited allowing the possibility of intentionally corrupting the system. The input data exposed in a URL can easily be modified by entering the changed URL directly in the location field of a web browser. The hidden fields inside an HTML form can easily be edited by saving the HTML file and then reloading an edited version of the web page. The problem with this is that any input that has not been properly parsed and filtered could gain access to the server by executing CGI scripts or other programs on the server. Server Side Includes also impose several problems. The server has to parse the HTML file, searching for SSI commands. After finding these commands, they have to be executed and their output merged into the HTML file. Since the HTML files can easily be edited, the SSI *exec* command can be used to execute any program on the server. This imposes an obvious security risk; but the web server software provides options to control the execution of SSI

commands. The second major problem is that CGI scripts are slow. CGI scripts are interpreted and run on the server line by line, so long or complicated scripts cause a performance problem and they do not scale well as the number of requests increases.

1.1.4 HyperText Markup Language

HyperText Markup Language (HTML) is the language of the web pages. HTML, a subset of the Standard Generalized Markup Language (SGML), provides a standardized method for formatting the contents of web pages. It is a collection of platform-independent style codes, indicated by markup tags, that defines the various components of a hypertext document [NCSA 97].

HTML documents are plain text files that are interpreted by web browsers and displayed as formatted web pages. The HTML tags direct the browser's efforts in formatting the document. An HTML file contains a set of elements that can contain plain text, other elements or both. An element is made up of two tags paired together. The starting tag, in the general form of *<tagname [attributes]>*, denotes the beginning of an element and the ending tag, in the form of *</tagname>*, denotes the ending of an element. Generally there are two types of markup tags; one set of tags defines the display characteristics of the document and another set defines the structure of the document and its interconnections to other documents on the web. The display related tags can be divided into character level tags and block level tags. For the character formatting tags, there are two types of tags, logical formatting style and physical formatting style. The logical formatting style provides a meaningful description of intent; for example, the

** tag is a means of drawing attention to the text inside this tag. Web browsers could display the text using a bold style, a bigger size font or maybe even a different color. But a physical formatting style, such as *<b>*, denotes the actual appearance of the text. A minimal HTML document is shown in Figure 5.

```
<html>
<head>
<TITLE>A Simple HTML Example</TITLE>
</head>
<body>
<H1>HTML is Easy To Learn</H1>
<P> Welcome to the world of HTML.
    This is the first paragraph.
    While short it is still a paragraph!</P>
<P> And this is the second paragraph.</P>
</body>
</html>
```

Figure 5: HTML Example. Source: *A Beginner's Guide to HTML.* http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimerAll.html. The National Center for Supercomputing Applications. 1997.

HTML tags generally are not case sensitive. So in the example above, Figure 5, tags that are in uppercase letters are examples of tags that define the display characteristics of the web page and tags that are in lowercase letters are examples of tags that define the structure of the web page.

1.1.4.1 Current Version 3.2: This sub-section describes the current version of HTML, version 3.2, which was finalized at the beginning of this year. This sub-section is organized into three parts. The first part contains text and graphics formatting information for the current version. The second part contains information about the lists,

tables and hypertext link capabilities of the current version. The last part describes advanced formatting features and capabilities, such as, frames, forms and image maps.

1.1.4.1.1 Text and Graphics Formatting: In the area of text formatting, there have been some improvements over the previous versions. HTML 3.2 provides tags for basic physical text formatting options, such as, bold (<*b*>), italics (<*i*>), type writer text (<*tt*>), underline (<*u*>), strikethrough (<*strike*>), subscript (<*sub*>), superscript (<*sup*>), smaller size (<*small*>), and bigger size texts (<*big*>). It also provides the logical text formatting options, such as, citation (<*cite*>), code (<*code*>), emphasis (<*em*>), keyboard entry (<*kbd*>), sample (<*samp*>), strong (<*strong*>), and variable (<*var*>). One of the biggest change in the area of text formatting is the addition of the <*font*> tag and the <*basefont*> tag [HONEYCUTT 97]. The <*font*> tag specifies the font type to be used for a given text and the <*basefont*> tag specifies the general font type to be used throughout the HTML file. In previous versions this was left up to the browser to implement; but now in HTML 3.2, the web page designer has control over the font used to display the text in their pages. But the major limitation with this method is that if the specified font is not available at the client location, it is still up to the browser to find a replacement font. A minor solution to this problem is the *face* attribute of the <*font*> tag; this attribute allows the author to specify alternate fonts to use. An example of the <*font*> tag is shown below; the syntax for <*basefont*> is also similar to this.

```
<font face="Arial", "Helvetica", "Times" size=12 color=blue>
```

In this example, the designer's primary choice for the font type is "Arial." The second and third choices are "Helvetica" and "Times," respectively. The font size and the text color are also specified in the <*font*> tag.

In the area of graphic formatting, little has changed from previous versions. Like the previous version, HTML 3.2 provides the <*img*> tag to display inline images. An example for this tag is shown below:

```
<img src="sample_1.gif" align=bottom align=left
                        vspace=20 hspace=20>
<img src="sample_2.gif" width=320 height=240 border=2>
```

In the first example, the "sample_1.gif" image is displayed, aligned to the bottom left relative to the surrounding text and a space of 20 pixels is given between the image and the surrounding text, both vertically and horizontally. In the second example, the "sample_2.gif" image is displayed with a 2 pixel border around it, and hints for the suggested size of the image are given in pixels.

1.1.4.1.2 Lists, Tables and Links: HTML 3.2 has made some changes in its support for lists, over previous versions. The lists created on web pages can come in several forms: numbered, bulleted, menu, directory, and definition lists. The major changes to HTML 3.2 in the area of lists, over previous versions, are the addition of several attributes to the <*ol*> tag and the <*ul*> tag [HONEYCUTT 97]. Additional attributes, such as, *compact*, *type*, and *start* allows control over the type of numbers used in an ordered list. Addition of the <*type*> attribute to the <*ul*> tag provides manual control over the type of bullets used in unordered lists. One minor change in the area of lists is the addition of the

compact attribute to the <*dl*> tag, which allows the creation of definition lists using a smaller font size.

The <*ol*> tag is used to create an ordered list and the <*ul*> tag is used to create an unordered list. The numbered list is a type of ordered list and the bulleted list is a type of unordered list. The menu lists and directory lists are similar to the unordered lists, the <*menu*> tag and the <*dir*> tag are primarily used for identification purposes. The definition list type, also called the glossary list, allows the creation of dictionary type listing with indented definition paragraphs. Examples for all three types of lists are shown in Appendix B.1.

The need for tables has existed almost since the beginning of the World Wide Web. Tables provide a natural method to display information such as comparative analysis and other tabular data. Tables can also be used to invisibly divide the web page into different sections for layout purposes. One possible use for this method would be in web pages that attempt to simulate print publications, as in multi-column newspaper format. HTML 3.2 finally provides official support for the creation of tables using HTML tags; but Netscape and Microsoft extensions have long been used to create tables on the web [HONEYCUTT 97]. The HTML 3.2 <*table*> tag along with the <*tr*>, <*td*>, <*th*>, and <*caption*> tags allows the creation of a table composed of rows and columns. Attributes such as *border*, *align*, *rowspan*, *colspan*, *width*, *height*, *cellpadding*, *cellspacing*, *bgcolor*, *bordercolor*, and *valign*, provide additional options to control the display characteristics of the table. An example of a table, with all these tags and attributes, is shown in Appendix B.2.

Links are the most important feature of the World Wide Web. The HTML anchor tag, *<a>*, is used to encode hypertext links in web pages. There have been no changes in the anchor tag, since the previous version of HTML. An example of a hypertext link using the anchor tag is shown below. The URL following the *href* attribute can be specified using either an absolute reference or a relative reference. The *<base>* tag can be used to fix the base, for relative references. A named bookmark can also be used to refer to subsections of an HTML document. An anchor tag example is shown below:

```
<a href=http://megahertz.njit.edu/~dnp3128/Thesis/LOGIN.HTM>
    REPI Web Site
</a>
```

1.1.4.1.3 Frames, Forms, and Imagemaps: The use of frames provides the web page designer with a flexibility that would not be possible by any other means. Even though frames have been supported since Netscape Navigator version 2.0, HTML 3.2 still does not support the creation of frames as a standard [HONEYCUTT 97]. But since the top two web browsers Netscape Navigator and Microsoft Internet Explorer, with a combined market share of over 90%, support frames and since the use of frames has become quite common on the World Wide Web, this section describes the *<frame>* tag and its associated tags and attributes.

The *<frameset>* tag creates independently controllable sub-windows, called frames, within the web browser's main window on the client machine [HONEYCUTT 97]. These frames provide the web designer with the ability to organize the web site more clearly and present navigation options in an easier to use manner. Each frame is in essence an independent mini-browser; in-fact an option in the *<frame>* tag provides for

the launching of another browser, completely independent from the previous instance of the browser. But one frame can control another frame within the same browser, including the ability to load new HTML files into another frame, or to create and destroy more frames within other frames. The major problem with the use of frames is that it does not degrade well in browsers that don not support Netscape's *<frame>* extension tag. At least the tables are displayed, although in a very unattractive manner, in browsers such as Lynx; but the frames can not even be displayed in such text based browsers. But as a solution to this problem, the *<noframes>* tag is provided, which allows the creation of content for text based browsers.

Frames have to be created using two sets of HTML files. The first set contains only one file, the frame creating file, also called the frame document. The second set contains one file for each frame created in the frame document. The frame creating file uses the *<frameset>* tag to create the frames and uses the *<frame>* tag to point to the HTML files that contain the contents of individual frames. An example of the frame document, "frames.htm," is shown in Appendix B.3, first figure. The *<frameset>* tag uses the *rows* and the *cols* attributes to divide the main window into four frames. First the browser window is divided into two frames, using the *cols* attribute, the left frame occupying 1/5th of the width and the right frame occupying 4/5th of the width. Next, the left frame is divided into two frames, using the *rows* attribute; the top frame occupies 10% of the height and the bottom frame occupies the remaining height of the browser window. Then the right frame is divided into two frames, again using the *rows* attribute; the top frame occupies 10% of the height and the bottom frame occupies the remaining height of the

browser window. Attributes for the *<frameset>* tag, such as, *frameborder*, *border*, and *bordercolor* allow control over how the borders between the frames are displayed. Using the *name* attribute, the top left frame is named "LOGO" and using the *src* attribute, its contents are linked to the "logo.htm" file. The top right frame is named "HEADER" and its contents are in the "header.htm" file. The bottom left frame is named "NAV_BAR" and its contents are in the "nav_bar.htm" file. The bottom right frame is named "MAIN" and its contents are in the "main.htm" file. Support for text based browsers is provided, using the *<noframes>* tag, by listing and providing a link to the HTML files that make up individual frames.

The second set of files is shown in the second and third figures of Appendix B.3. Each file, in the second figure of Appendix B.3, is separated using the HTML comment tags (<-- and -->). The first file, "logo.htm," simply loads an image file as the logo. The second file, "header.htm," displays a title for this page. The third file, "nav_bar.htm" provides a set of links to open the individual frames. The second figure of Appendix B.3 shows the fourth file, "main.htm," which provides information about this web page. The links in the navigation bar frame show several methods of controlling an HTML page with multiple frames in it. The "Logo" link clears the current browser window and displays the "logo.htm" file. The "Header" link destroys the right two frames and displays the "header.htm" file. The "Navigation Bar" link shows the effect of targeted frame loading; it displays a navigation menu in the bottom right frame. The "Main" link opens up another instance of the browser to display the "main.htm" file.

Forms are the main method of gathering user input and providing interaction between the web user and the web site. Without forms, the World Wide Web can only be used as a publishing system instead of as a "platform" for application development. HTML 3.2 provides increased support for forms when compared with previous versions. The *<form>* tag is used to create an input form on the web page and to provide connections to the server side programs to process the entered input. Additional tags, such as *<textarea>*, *<select>*, and *<input>* allow the creation of various types of data fields on the form. The *<textarea>* tag provides for a free-form text entry field. The *<select>* tag provides for the creation of a select scroll box or the drop-down list. The *<input>* tag can be used to create various types of data controls using the type attribute.

Forms support is one of the first steps in expanding the role of HTML and the World Wide Web. HTML can use forms to create interactive web pages and through interaction World Wide Web can be used as more than just a publishing system. But forms have several limitations in them. A major limitation is that HTML forms have no intelligence of their own. All forms data have to be sent to the server for even the simplest of error checking, such as checking to see if all the data fields are filled in. A simple addition of a "required" attribute to a data field would save one trip to the server in many cases. Another limitation is that HTML forms are very primitive compared to the features and control available in a graphical platform such as Microsoft Windows. Forms are not displayed consistently across platforms because the browser for a given graphical platform uses the native graphical widgets and controls to display the HTML

form widgets and controls. So the "look and feel" of the HTML form can not be maintained across platforms.

Appendix B.4 shows an example of a form with various data fields and controls in it. The example form has four text input fields, one password field, two check box controls, a group of radio option controls with three buttons in the group, a multi-select scroll box, a drop down list box, and a text area control. It also has a submit command button and a reset command button. The submit command button uses the *post* method to send the form data to the server, where the "form.cgi" script will be executed to process the forms data. The reset command button will clear all the data fields and re-display the form.

Imagemaps are ordinary graphic images, on which different areas can be linked to different resources on the Internet. Imagemaps can be used as a graphical menu, a more attractive alternative to a simple text based menu. Imagemaps are the natural choice for applications with links that have a spatial relation to each other [HONEYCUTT 97]. An example would be to use a map of the world or a country to provide links to regional information. The major disadvantage is that imagemaps are limited to graphical browsers and even then image load can be turned off by users using low bandwidth connections.

Imagemaps can be created on the server side or on the client side. HTML 3.2 improves the use of imagemaps by providing official support for client-side imagemaps. The basic difference is the location of the mapping coordinates for the hot spots. A hot spot is a well-defined area on the imagemap which is linked to a URL. In server-side imagemaps, the browser sends the coordinates of the mouse click to the server. The

server then uses a map file to look up the target address and retrieves the target, if the clicked coordinates are inside a hotspot. The major limitation with this method is that a trip to the server is needed just to find the target location. An improvement on this is that the hot spot area coordinates can be included directly in the HTML file containing an image map, by using the *<MAP>* and *<AREA>* tags along with the *usemap* attribute of the *<IMG>* tag. This method of imagemaps are called client-side imagemaps [HERRMANN 96]. In the client-side imagemaps, the client browser uses the map coordinates from the *<AREA>* tag to find the target URL if the clicked coordinates are inside a hotspot. The client then sends a HTTP request header using the target URL.

Client side imagemaps and server side imagemaps both have advantages and disadvantages. Browsers can use client side imagemaps to provide immediate feedback to the user, as the user moves the mouse cursor over an imagemap. For example, the target URL can be displayed on the status line of the browser as the mouse cursor moves from one area on the map to another area. The disadvantage is that map area coordinates have to be maintained in the HTML file. Server side imagemaps use a separate map file to store the map area coordinates, but each click on an imagemap requires a trip to the server.

```
<html>
<head><title>HTML <em>Imagemap</em> Example</title></head>
<body><map name=menumap>
<area shape=rect coords="0,0,100,100" href=rectangle.htm>
<area shape=circle coords="110,110,5" href=circle.htm>
<area shape=poly coords="150,150,200,200" href=polygon.htm>
<area shape=default href=noref></map>
<center><img src="menumap.gif" usemap=#menumap></center>
</body>
</html>
```

Figure 6: HTML Imagemap Example

An example of a client-side imagemap is shown in Figure 6. In this imagemap, four hot spots are defined. The first area defined is a rectangle, from coordinates "0,0" to "100,100," which points to the "rectangle.htm" file. The second area is a circle, with a radius of 5 centered at "110,110," which points to the "circle.htm" file. The last defined area is a polygon, from coordinates "150,150" to "200,200," which points to the "polygon.htm" file. If the clicked coordinates are outside these areas, a default area is defined that does not refer to any location.

1.1.4.2 HTML 4.0: HTML was originally designed as a standard for representing text and images on web pages. But with the increased popularity and availability of the World Wide Web, HTML is increasingly being used for more powerful applications. HTML and web pages created from them are becoming a universal platform-independent client interface, used as a "universal desktop" for a broad range of applications running on the Internet application servers [NETSCAPE 96]. Netscape's white paper on their vision for future network centric applications states that HTML could be thought of as a universal "resource definition language," similar to resource files found in traditional programming environments. "HTML truly blends content and applications until the two become indistinguishable - in effect, the content *is* the application" [NETSCAPE 96]. To support this increased responsibility, HTML has to evolve to provide the necessary functionality.

In the past the World Wide Web Consortium (W3C), an organization that directs the future of the web, had adapted many of the advances made by Netscape as standard tags in their latter versions of the HTML standard. The support for forms is one of the

examples for past versions of the HTML standard, and the support for tables in HTML version 3.2 is a recent example. But with the recent popularity of Microsoft Internet Explorer, Microsoft has gained enough market share to make strong advances of their own. Now W3C's role has changed from merely adopting advances made by Netscape, into acting as an arbitrator between the advances made by Netscape and Microsoft [ZGODZINSKI 97].

For the next version of the HTML standard, HTML 4.0, Dynamic HTML represents the advances made by Netscape and Microsoft that have to be adapted as standard. HTML has been evolving in two general areas: increased power and more flexibility. For the past versions of the standard, forms, imagemaps, and tables represented the increase in power and flexibility. For the next version, Dynamic HTML represents the increase in power and style sheets represent the increase in flexibility. The general direction and the basic idea behind HTML is to separate the contents of a page from its presentation style. W3C's idea is to stop using HTML tags to define layout and presentation issues. HTML has had two types of character formatting, logical and physical, since its earliest version. The separation of logical and physical styles was an early effort at separating the contents and the presentation style. The World Wide Web Consortium's Cascading Style Sheets (CSS) specification is the recent step taken in separating content from presentation.

1.1.4.2.1 Cascading Style Sheets: Cascading Style Sheets is a mechanism to attach styles to HTML elements such as *<body>*, *<P>*, etc. CSS is set of rules that consists of an HTML selector and style declarations. A simple CSS rule is shown here: *H1 {color:*

blue}. In this example, the HTML selector is the level one heading tag, *<H1>*, and the style declaration is the value "blue" set to the *color* attribute of the *<H1>* tag. After this style has been defined, all level one headings in this HTML file will be displayed in blue color.

```
<html>
<head>
    <title>Title</title>
    <link rel=stylesheet type="text/css"
         href="http://style.com/cool.css" title="Cool">
    <style type="text/css">
        @import url(http://style.com/basic);
        H1 {color: blue}
    </style>
</head>
<body>
    <H1>Headline is blue</H1>
    <P STYLE="color: green">
        While the paragraph is green.
    </P>
</body>
</html>
```

Figure 7: CSS Example Part 1 of 2. Source: Wium Lie, Håkon and Bos, Bert. *Cascading Style Sheets, level 1*. W3C Recommendation REC-CSS1-961217. http://www.w3.org/pub/WWW/TR/REC-CSS1. World Wide Web Consortium. 1996.

This simple idea of attaching styles to HTML elements can be extended in many forms. Style sheets come in three varieties: linked style sheets, embedded style sheets, and inline style sheets. The different possible methods of using CSS are shown in Figure 7. In this example, the "Cool" style is linked from a separate style sheet, stored at a location referred to by the URL specified in the *href* attribute. The embedded type of style sheets is shown in two examples; the imported example and the attached example. The "basic" style sheet is imported from the given URL. A style declaration is attached

to the *<H1>* tag. The inline type of style sheets is shown as the *<P>* tag is modified by adding the CSS *style* attribute.

Netscape's version of Cascading Style Sheets is implemented using JavaScript and the proprietary *<layer>* tag. JavaScript is described in more detail in Section 1.3.2. Netscape's version of CSS is called by various names: JavaScript Style Sheets (JSSS), JavaScript Accessible Style Sheets (JASS), JavaScript based Style Sheets, and Dynamic Style Sheets. JavaScript Style Sheets technology is a non-standard method not approved by the W3C. Since JSSS is incompatible with W3C's CSS and it is at odds with the future of HTML and other web publishing technologies, it is not focused on too much.

```
<Style type="text/css">
    P   { color: blue, font: italic; text-align: center}
</style>

<style type="text/css">
    .IMPORTANT  { color: red; text-decoration: underline; }
</style>

<P class=IMPORTANT>
    An example of the important text, which will be
    displayed using a red underlined characters.
</P>
```

Figure 8: CSS Example Part 2 of 2. Source: Spelman, Jennifer and Rein, Lisa. "CSS or JSS: Which will better suit your needs?" http://www.netscapeworld.com/netscapeworld/nw-07-1997/nw-07-css.html. NetscapeWorld. 1997.

Examples of CSS and JSSS notations for style definition and class definition are shown in Figures 8 and 9, respectively. Both examples define a similar style for the HTML tag *<P>* and create a logical style class called "IMPORTANT" with a set of display properties. An example of how to use the logical style in an HTML file is also

shown in Figure 8. As can be seen from these examples CSS and JSSS are more similar than they are different.

```
<Style type="text/javascript">
    tags.P.color = "blue";
    tags.P.fontStyle = "italic";
    tags.P.lineHeight = "1.5";
    tags.textDecoration = "capitalize";
    tags.textAlign = "center";
</style>

<style type="text/javascript">
    classes.IMPORTANT.all.color = "red"
    classes.IMPORTANT.all.textDecoration = "underline"
</style>

<P class=IMPORTANT>
    An example of the important text, which will be
    displayed using a red underlined characters.
</P>
```

Figure 9: JSSS Example. Source: Spelman, Jennifer and Rein, Lisa. "CSS or JSS: Which will better suit your needs?" http://www.netscapeworld.com/netscapeworld/nw-07-1997/nw-07-css.html. NetscapeWorld. 1997.

There are many advantages in using CSS to define layout and presentation instead of using HTML tags for those purposes. The separation of content from presentation provides the key benefit of cross-platform uniformity. Style sheets can be used at platform level to get an exact match across platforms, and HTML files can be used as just a container for the document's contents including scripts.

The second major advantage is that Cascading Style Sheets are designed to gracefully degrade [SPELMAN 97]. CSS uses a hierarchy of formatting levels. At the highest level, global style sheets are used to define a consistent "look and feel" for the whole web site. At the next lower level individual style sheets can be used, linked to either a small subset of pages or even individual pages. At an even lower level, CSS

formatting attributes can be used inside individual HTML tags. At the lowest level, individual users can use their own style sheets to define their custom "look and feel." Another way of looking at this same advantage is to think of it as providing inheritance and aggregation control for formatting HTML elements. That is, a generic style can be defined at the <*body*> tag level and all tags inside the <*body*> tag inherit the style defined at the <*body*> level [SPELMAN 97].

CSS also provides support for people with disabilities. For example, a visually impaired user can use customized style sheets that use audio to "display" the contents of a web page stored in a generic HTML file. Other forms of disabilities can be overcome by using different types of style sheets to "display" the contents in a manner that is accessible to those users.

"Another advantage is that separating content from presentation allows authors to be authors and designers to be designers" [REIN 97]. A content creator can create the web page's contents and a designer can design the web pages without interaction or dependencies between each other.

W3C's draft on Positioning using Cascading Style Sheets (CSS-P) provides a superset of functionality over the <*layer*> positioning technique and it also provides the flexibility, power and ease of use provided by CSS [REIN 97]. For example, the CSS-P notation for positioning items is shown in Figure 10 with the script code to dynamically change its properties.

```
<img name="image_name" style="position: absolute; top:10px;
                              left:25px" src="css_p.gif">
document.images["image_name"].style.top = "15px"
document.images["image_name"].style.left = "30px"
document.images["image_name"].src = "2nd_img.gif"
```

Figure 10: CSS-P Example

But Netscape's notation for the same positioning requires the use of *<layer>* tag to surround the *<img>* tag. An example for the *<layer>* notation is given in Figure 11 with the JavaScript code to dynamically change its properties. This not only creates extra unnecessary lines, but its usage for manipulation is also unnatural as can be seen from mixed thinking required for manipulating the properties of *<layer>* and its associated *<img>*.

```
<layer top=10 left=25 name="layer_name">
    <img src="layer.gif" name="image_name">
</layer>
document.layers["layer_name"].top = 15
document.layers["layer_name"].left = 30
document.images["image_name"].src = "2nd_img.gif"
```

Figure 11: JASS Example

CSS-P's other advantage is that it is fully compatible with the CSS concept of linked style sheets and hierarchical formatting control. As an example, the common need to position a company's logo consistently on all the pages in a web site, is used to illustrate the disadvantages of the Netscape's technology in comparison with W3C's CSS-P technology. Using Netscape's *<layer>* notation, every page on the web site requires an extra set of tags and specification to position the logo. This creates

maintenance problems in any web site that has more than a few pages. Using W3C's approach one style sheet can be used to define the logo's position and other characteristics. Every page on the web site uses this style sheet to position the company's logo on that web page. This method requires maintenance on only one file, the style sheet [REIN 97]. Another limitation of the Netscape method is that the *<layer>* tag only supports the use of pixels and percentages when positioning HTML elements. CSS and CSS-P supports many other methods for representing positions, including pixels, percentages, points, ems, etc. Another limitation of the Netscape's JSSS is that, as the name suggests, it requires JavaScript; but CSS can be used with any scripting language [SPELMAN 97].

Taking these issues into account, Spelman and Rein wrote "JSSS is harder to use than CSS, but offers a superset of capabilities," because of the power of JavaScript [SPELMAN 97]. Their article comparing the pros and cons of CSS vs. JSSS concludes that CSS should be used unless the extra dynamic capability is needed at the cost of extra work and loss of compatibility.

1.1.4.2.2 Dynamic HTML: Today's HTML based web pages are static and non-interactive by themselves. Additional support such as CGI programming or Java programming is needed to dynamically change the contents of web pages or to provide any level of interaction for the user. Both methods require a round trip to the web server before the contents of the page can be changed. This method of changing the page's contents or providing interaction is not only less responsive, it also increases network

traffic and server load. The answer to these problems is to provide more control of the HTML web page to the browsers allowing them to make changes to the web page without requiring a round trip to the server. "Dynamic HTML adds richer, more engaging user interfaces to the HTML presentation language, while also greatly reducing the workload on networks and servers" [MICROSOFT 97a].

Netscape uses JASS, layers, and dynamic fonts to provide the functionality of Dynamic HTML. JASS allows the creation of style sheets using JavaScript. But JASS only provides dynamic abilities while the browser is parsing and rendering a web page. Once the web page has been loaded, it can not be altered using JASS. Layers technology allows web designers to position page elements using the *<layer>* and *<ilayer>* tags. But this technology combines the presentation of a page to its contents; this is incompatible with W3C CSS which separates the presentation of the page from its contents. But according to Netscape, layers technology builds on existing standards by adding dynamic extensions, with a tag, to existing HTML scripts [ZGODZINSKI 97].

Microsoft's version of the Dynamic HTML provides an object model for the web page [MICROSOFT 97a]. The web page becomes an object that can be manipulated using scripts or programs. It also includes multimedia and database features. Database features allow easier creation of dynamic forms for applications such as master-detail order entry. In applications such as these, sorting or recalculations can be applied without a trip to the server. Multimedia features allow movements of graphics in 2 dimensional plane or in the 3 dimensional space.

Dynamic HTML is compatible with the W3C CSS specification, which makes it compatible with current browsers and existing HTML pages. The Dynamic HTML Object Model does not add new tags to HTML; it makes existing tags, attributes, and CSS attributes programmable using any type of scripts, controls or applets [CLUTS 97]. Dynamic HTML provides access to all the elements on the page, exposing their properties, methods, and events. An example of this is shown in Figure 12. In this example, the level one heading tag has been extended with two events, "onmouseover" and "onmouseout," and a style sheet has been added to the page. This example changes the level one heading into a red color when the mouse cursor moves over the heading text and changes it to blue color when the mouse cursor moves away from the heading text.

```
<style>
     .redText {color:Red}
     .blueText {color:Blue}
</style>

<H1>
     onmouseover = "this.className = 'redText' "
     onmouseout = "this.className = 'blueText' "

     Make this text red
</H1>
```

Figure 12: Dynamic HTML Example. Source: Cluts, Nancy Winnick. *The Dynamic HTML Object Model.* http://www.microsoft.com/intdev/ie4/omdoc-f.htm. Microsoft Corporation. April 1997.

Microsoft has published a report that compares Netscape's version of Dynamic HTML with its own version [MICROSOFT 97b]. In general this report, claims that Microsoft's version is more compatible with W3C specifications and that it provides greater control and flexibility for web page designers. More specifically, this report says

that Microsoft's Dynamic HTML has a more comprehensive object model exposing more elements on a web page that can be programmed with more language options than Netscape's Dynamic HTML. Microsoft's Dynamic HTML has the ability to be changed after load time, whereas Netscape's version can only be changed at or before load time. Microsoft's version provides more control in positioning page elements and allows full animation of those elements. Microsoft's Dynamic HTML allows data binding from different sources whereas Netscape does not provide any database features in its Dynamic HTML. Netscape uses Java multimedia classes to provide multimedia support, Microsoft uses HTML and scripts to provide the same support. Microsoft claims that their Dynamic HTML provides an easier method to make pages interactive and that its technology is based on open standards.

1.2 Messaging Protocols and Standards

One of the earliest applications on the Internet was the electronic mail, which provided the ability for people to exchange messages between one computer and another computer. These electronic messages are exchanged between computers using several protocols, such as Simple Mail Transport Protocol (SMTP), Post Office Protocol (POP), Internet Mail Access Protocol (IMAP). Until recently these electronic messages, or email as it is commonly called, were limited to text only messages without any formatting attributes other than spaces and tabs. But now email clients provide the ability to send formatted messages using HTML. Besides formatted text messages, email can now include just

about any type of content using the Internet Media Type standard, formerly called Multipurpose Internet Mail Extensions (MIME).

This part of the chapter first describes the mail access paradigms. After that, SMTP is described, which is a protocol for sending email messages on the Internet. Next the POP and IMAP protocols are described and compared. These two protocols provide for the ability to remotely access email from client machines. Finally the Internet Media Type standard is described, which provides a standard method for describing an email message's contents.

A client machine on the Internet has to connect to a mail server to access the user's email messages. There are three mail access paradigms: Off-line, Disconnected, and On-line [GRAY 95a]. In "off-line" mode, the client software completely downloads the contents of a mailbox and removes it from the server. The user is then allowed to read and manipulate the messages and folders on the client side using local storage. Any changes made on the client side are not reflected on the server side. The major disadvantage for this mode is that it does not give access to a user's messages from more than one client machine. If a user accesses email from more than one client using the "off-line" method then some of the messages will be stored in one computer and other messages will be stored in other computers. "Disconnected" mode and "on-line" mode provide different levels of support for this common situation, where access to the common mailbox is needed from more than one location. In "disconnected" mode the client software connects, downloads the contents and disconnects. The messages are still left intact on the server. After the user reads and manipulates the messages on the client

side using local storage, the client software re-connects to the server and updates the server's state. This provides for people who access their email from different computers. In "on-line" mode, the client software is in continuous contact with the server and any changes made to the messages or the folders happen directly on the server. This also supports people who want location independent access to their email; but this mode's extra cost is in its need for a continuous connection to the server.

In summary, the "off-line" mode provides on-demand access to a single client while the "on-line" mode provides access to multiple clients. "Off-line" mode minimizes connect time and server requirements; while "on-line" mode needs longer connect time, bigger server storage and more processing power. "On-line" mode does not maintain any state information on the client side, which makes it attractive for diskless clients or for "public" client machines such as computer labs. "On-line" mode also provides for the possibility of shared mailboxes [GRAY 95b].

1.2.1 Simple Mail Transfer Protocol

Simple Mail Transfer Protocol (SMTP), as the name suggests, is simply meant for transferring mail from one computer to another computer. For SMTP to reliably and efficiently transfer the electronic mail, a transmission subsystem providing a reliable ordered data stream channel is needed. SMTP is not dependent on any particular transmission protocols. SMTP can work across transport service environments by using a common inter-process communication environment [RFC821].

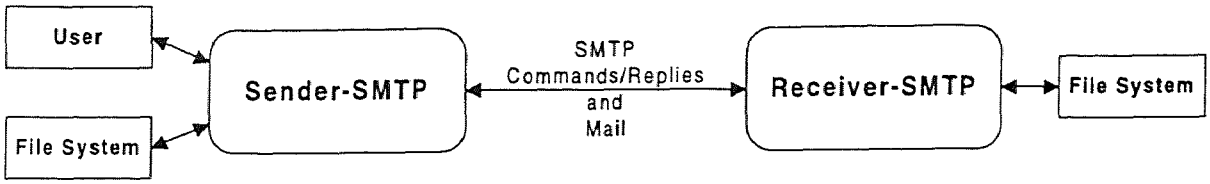


Figure 13: SMTP Model. Source: Postel, Jonathan B. "Simple Mail Transfer Protocol." Request for Comment (RFC) 821. http://ds.internic.net/rfc/rfc821.txt. 1982.

Figure 13 shows an overview of the SMTP model, described here. The SMTP design provides for a sender to communicate and send commands to a receiver, which receives and responds to the given commands. A SMTP session is as follows: *user_A* on *host_1* requests to send mail to another *user_B* on *host_2*. The sender-SMTP establishes a two-way transmission channel to a receiver-SMTP; the receiver-SMTP might be the final destination *host_2* or it might be an intermediate node on the network. The sender-SMTP uses the *MAIL* command to indicate the sender of the mail and the *RCPT* command to indicate the receiver of the mail. SMTP-receiver is not required to accept arbitrary recipient addresses; but some servers accept arbitrary addresses and forward them to another host, forming a relay chain from the sender to the final destination [RFC1939]. After each step, the receiver-SMTP has to acknowledge the command or reject it. After the *RCPT* command, the receiver-SMTP might reject the recipient but not the transaction, indicating that this receiver is just an intermediary. After the source and the destination have been identified, the message text and headers are sent using the *DATA* command. An example of this session is shown in Figure 14. In this example, email headers are not explicitly sent by the client after the *DATA* command; so the

sender-SMTP automatically generates the required minimum headers such as "Date," "From," "Message-Id," and "Status."

```
Server> 220 megahertz.njit.edu ESMTP Sendmail 8.8.5/8.6.9
        ready at Wed, 13 Aug 1997 12:14:57 -0400 (EDT)
Client> HELO megahertz.njit.edu
Server> 250 megahertz.njit.edu Hello megahertz.njit.edu
        [128.235.251.100], pleased to meet you
Client> MAIL FROM:<dnp3128>
Server> 250 <dnp3128>... Sender ok
Client> RCPT TO:<dnp3128@megahertz.njit.edu>
Server> 250 <dnp3128@megahertz.njit.edu>... Recipient ok
Client> DATA
Server> 354 Enter mail, end with "." on a line by itself
Client> <Email message body sent here>
Client> .
Server> 250 MAA05271 Message accepted for delivery
Client> QUIT
Server> 221 megahertz.njit.edu closing connection
```

Figure 14: SMTP Example

Besides the commands shown in the above session dialogue, Figure 14, other commands are supported by SMTP. These commands provide for mailing list expansion (*EXPN*) forwarding, user lookup (*VRFY*), and sending to a group list. In addition to these commands and the MAIL method of mailing email, SMTP provides for three other methods of sending mail. SMTP distinguishes between two types of email delivery: "sending" and "posting." An email is "posted" to a user's mailbox; on the other hand an email is "sent" to the user's terminal. The "sending" method of email delivery is supported by using the commands: *SEND*, *SOML*, and *SAML*. However, these commands are optional and are not required to meet the minimum implementation standard [RFC821]. Besides the standard SMTP, several service extensions are defined in [RFC1869]; SMTP servers with these extensions are identified by "ESMTP" in their

initial response and ESMTP also responds to the initial *EHLO* command instead of the *HELO* command. An example session with an ESMTP server is shown in Appendix C.1. As part of their response to an *EHLO* command, they list all the extensions that they support. If a client issues a *HELO* command to an ESMTP server then the regular response is given as defined in [RFC821]. In Appendix C.1, the *HELP* command is used to show a listing of the new commands supported by ESMTP.

The format for the email's headers is documented in [RFC822]. This document specifies the syntax of Internet mail headers. The headers section of an email consists of several header lines followed by a single blank line. Each individual header can be multi-lined with the second and subsequent lines indented with whitespace. Headers consist of a field name, a colon, and a field data. Whitespaces are not allowed before the colon, and field data has additional syntax requirements for certain fields.

```
Received: from njit.edu (homer.njit.edu [128.235.35.70]) by
    megahertz.njit.edu (8.8.5/8.6.9) with SMTP id OAA27585
    for <dnp3128@megahertz>; Mon, 28 Jul 1997 14:52:19 -0400
    (EDT)
Received: from ocean.njit.edu by njit.edu (5.x/SMI-SVR4)
    id AA08402; Mon, 28 Jul 1997 14:52:43 -0400
Received: by ocean.njit.edu (SMI-8.6/SMI-SVR4)
    id OAA02495; Mon, 28 Jul 1997 14:39:11 -0400
Date: Mon, 28 Jul 1997 14:39:11 -0400
From: tanik@homer.njit.edu (Dr. Murat Tanik)
Message-Id: <199707281839.OAA02495@ocean.njit.edu>
To: dnp3128@megahertz.njit.edu
Subject: Re: Thesis Draft for Section 2.1
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
Content-Md5: pSUkOef4WsuPnBLxJDLLcA==
Status: RO
```

Figure 15: Email Header Example

An example for the email header section is shown in Figure 15. This example shows 13 header lines. The multi-lined "Received:" lines describe the route taken by the email, including the timing information for each node. The "From" header and the "To" header contains the email address for the message's source and destination, respectively. The general format of the email address is user@host.network, where the *user* field can either be one word or in a format such as "lastname.firstname." The *host.network* section provides the host name and the network name for the mail server. The "Message-Id" header provides a unique identification for this message; the general format for this consists of the date and time of the message, followed by the process id for the mail server, and the host name and network name for the mail server. The "Subject" header has no format requirements for its field data. The next four header lines are described in Section 1.2.5, titled "Internet Media Type." The "Status" header contains informational attributes about the email, such as: read, opened, deleted, etc.

1.2.2 Post Office Protocol

The Post Office Protocol - Version 3 (POP3), is used to allow client machines to remotely access the email stored on a mail server [RFC1939]. The important point about the POP3 is that it is only used for message retrieval; sending email messages is still dependent on SMTP. POP3 is not intended to provide extensive manipulation operations because POP3 clients are supposed to download all the messages and manipulate them on the client side. A POP3 session starts with client software establishing a connection to the server software. The POP3 server responds with a greeting, and then an exchange of

commands and responses is initiated between the client and the server, until the client closes the connection or the connection is terminated by an external event.

```
Server> +OK UCB Pop server (version 1.831beta) at
            megahertz.njit.edu starting.
Client> USER dnp3128
Server> +OK Password required for dnp3128.
Client> PASS ******
Server> +OK dnp3128 has 2 message(s) (1632 octets).
Client> STAT
Server> +OK 2 1632
Client> LIST
Server> +OK 2 messages (1632 octets)
Server> 1 990
Server> 2 642
Server> .
Client> LIST 3
Server> -ERR Message 3 does not exist
Client> RETR 1
Server> +OK 990 octets
Server> <The POP3 server sends the entire message here>
Server> .
Client> DELE 1
Server> +OK Message 1 has been deleted.
Client> STAT
Server> +OK 1 642
Client> RSET
Server> +OK 2 messages (1632 octets)
Client> STAT
Server> +OK 2 1632
Client> QUIT
Server> +OK Pop server at megahertz.njit.edu signing off.
```

Figure 16: POP3 Dialogue Example

An example of a POP3 session is shown in Figure 16. Upon receiving the greetings, the client sends the *userid* and *password* using the *USER/PASS* pair of commands. The server accepts the authorization information and sends an acknowledgment. The client then requests a status of the mailbox using the *STAT* command. The server responds with the information of two emails with a total size of

1632 octets. All multi-line responses from the server are terminated with a dot in the ending line. The client requests a *LIST* and the server responds with the information of the first message being 990 octets and the second message being 642 octets. The client requests a *LIST*ing of the third message, for which the server responds with a negative acknowledgment, *-ERR*. The client *RETR*ieves the first message and the server sends the contents of the first message. The client *DELE*tes the first message and requests a *STAT*us. The server responds with the information that there is one message with a total size of 642 octets. The client resets the mailbox, using the *RSET* command and requests another *STAT*us. The server responds with the original information of two messages totaling 1632 octets. The client issues the *QUIT* command and the server responds with a positive acknowledgment. Some other commands supported by POP3 are not shown in this example. The TOP command provides for the ability to retrieve only the top n lines specified in the command. The UIDL command allows the client to retrieve a unique identification for any of the email messages stored in the mailbox. The *APOP* command provides for an alternate way for the authorization function of the *USER/PASS* command pair. This command encodes the password information instead of transmitting it as plain text, as in the *PASS* command.

A POP3 session is a progression through different states between the client and the server and these states are *AUTHORIZATION*, *TRANSACTION*, and *UPDATE* [RFC1939]. A state diagram is shown in Figure 17, showing the progressions that are possible in a POP3 session. In the *AUTHORIZATION* state, the client identifies the user to the server. Upon a successful authorization, the server locks the user's mailbox and

the client initiates a number of transactions in the *TRANSACTION* state. After all the transactions are complete, the client closes the connection and the server enters the *UPDATE* state to commit the client's transactions and release the mailbox.

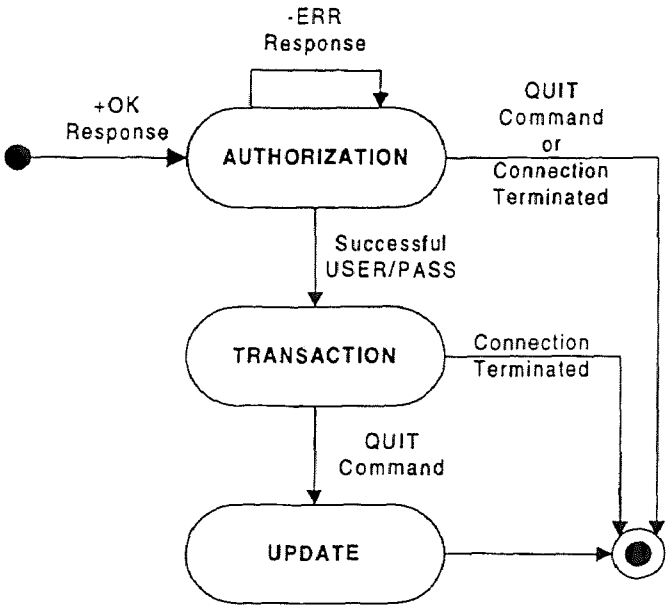


Figure 17: POP3 Session State Diagram

1.2.3 Internet Message Access Protocol

The Internet Message Access Protocol (IMAP), in its current version 4 (IMAP4), is an alternative for the POP method for remote mail access. IMAP4rev1, the latest revision of IMAP4, allows a client to access and manipulate messages in a remote folder on the server, as if these folders were stored on the local machine [IMAP4rev1]. IMAP supports all three mail access paradigms; thus making POP and Distributed Mail System Protocol (DMSP) no longer necessary for "off-line" and "disconnected" modes, respectively. IMAP, which once stood for "Interactive Mail Access Protocol," is really designed for interactive access to a common mailbox from anywhere on the Internet. The user reads

and manipulates their email messages directly on the server instead of on the client as in the case of POP. "IMAP is designed to permit manipulation of remote mailboxes as if they were local" [GRAY 95b].

An IMAP4rev1 session is a progression through many states, just as the POP3 session. In IMAP4rev1 the sessions are "Initial Connection," "Non-Authenticated," "Authenticated," "Selected," and "Logout." Appendix D.1 shows the state transitions as defined in [IMAP4rev1]. The "Initial Connection" state is the state after the client connects to the server. The "Non-Authenticated" state is the state during the login process. The "Authenticated" state is the state after the login process, either from an explicit login or from a pre-authenticated connection. After a mailbox has been selected, the session is in the "Selected" state. Just before the connection is closed, the session is in the "Logout" state.

IMAP4rev1 provides for message attributes that can be manipulated by the user. These message attributes come in two types: system flags and user-defined flags. The system defined flags are "\Seen," which indicates that a message has been read, "\Answered" which indicates that a reply has been sent for this message, "\Flagged," a generic indicator drawing special attention to this message, "\Deleted," indicating that this message is marked for deletion, "\Draft," indicating that this message's composition has not been completed, and "\Recent." The "\Recent" is a special system flag that can not be set by the user, it is automatically set by the server when new messages are received by it and the user has not yet "\Seen" it. In addition to these system defined flags, users can create and maintain their own flags. Both types of flags can either be

session flags, valid only for the current session or permanent flags, persistent across sessions.

IMAP4rev1 provides for complete control over remote mailboxes, including commands for creating, deleting, and renaming mailboxes, also called folders [IMAP4rev1]. It also has operations for manipulating messages with commands such as *APPEND*, *COPY*, *EXPUNGE*, *FETCH*, and *LIST*. IMAP4rev1 supports server side searching, with the *SEARCH* command; status checking, with *EXAMINE*, *SELECT*, and *STATUS* commands; and status flag maintenance, with the STORE command. The *FETCH* command can also be used to retrieve message structures, parts of messages or the full message. IMAP4rev1, like POP3, depends on SMTP to send messages.

In IMAP4rev1, client commands are tagged with a command identifier such as "A001." Server responses can either be tagged or untagged; tagged responses provide a direct response to the corresponding tagged command from the client and untagged responses provide additional data requested by client commands. Appendix D.2 shows an example for the IMAP4rev1 session. Like the POP3 session, an IMAP4rev1 session starts out with a login command. The *LOGIN* command can be used to login to the mail server. But like the *USER/PASS* command pair in POP3, *LOGIN* command sends the password openly. An *AUTHENTICATE* command is provided for coded transmission of password and other data. The *CAPABILITY* command is used for a negotiation of encoding schemes and other features supported by the server.

The client-server dialogue in an IMAP4rev1 session consists of a series of commands sent by the client, for which the server responds with data, possibly multi-line,

sent to the client. An IMAP4rev1 server could send status updates to a client, without an explicit request by the client. In the example session, shown in Appendix D.2, the *SELECT* command is used to select a mailbox to work with; this puts IMAP4rev1 session into "Selected State." The server responds with status information about the selected mailbox including such information as the number of messages and recent messages, allowed flags, read/write permissions, etc.. Then the *SEARCH* command is used to get information about new messages in the user's inbox. The *FETCH* command is used to retrieve the "Date" and "From" headers and the status flags for messages 2 to 4. The first message's status flag is updated using the *STORE* command and the *EXPUNGE* command is used to delete those messages. The first message is copied to the "Junk" folder using the *COPY* command. Finally the session is closed with the *LOGOUT* command.

1.2.4 POP3 vs. IMAP4

IMAP4 is similar to POP3 in many ways [GRAY 95a]. Both are mail access only protocols, relying on SMTP for email delivery. Both require a mail server to be "always up" receiving email and storing them in a user's "inbox." Both protocols are open, platform independent, Internet standards that can be used from anywhere on the Internet without requiring any email gateways. Both support message identifiers that are unique across sessions. But the similarities end here; IMAP provides more features, which makes it a better choice for most applications. The only situation where POP would be a better choice than IMAP is when server resources are limited or connection time needs to

be short. POP does not maintain a user's email on the server and no processing is done on the server. IMAP stores both new and old email for all the users on the server and email header parsing, searching and selection is done on the server. POP allows users to quickly download all their messages and work on them off-line, while IMAP requires a continuous connection while the user works on their messages. Using IMAP in "off-line" mode requires the same amount of server and connection resources as using POP. POP could be used in such a manner that quasi-online mode is possible [RFC1939]. POP3's *UIDL* command or the *RETR* command could be used without using the DELE command. This will allow clients to retrieve the messages and leave the original on the server. To prevent old messages as showing up as new messages, some state information has to be maintained by the client. A side effect of this type of use is that old messages accumulate on the server.

IMAP has many advantages over POP [GRAY 95a], [GRAY 95b]. Gray's article on remote mail access concludes that "supporting online and disconnected access is essential for a growing fraction of mail users and POP is inadequate for proper online and disconnected support, whereas IMAP does a good job for all three messages access paradigms" [GRAY 95a]. IMAP's advantages are in the areas of improved support for message management and improved performance for on-line connections.

IMAP includes the ability to set standard and user-defined message status flags. Remote folder and multiple folder manipulation provides for organizing messages on the server. Multiple folders, including a hierarchy of folders, can be created on the server, or multiple servers, to store and organize all the email messages in a user's mailbox. It also

supports simultaneous update and update discovery in shared folders [GRAY 95a]. IMAP can be used to access non-email information such as usenet messages and other documents. This capability can be used to unify different categories and sources of information [GRAY 95a].

IMAP servers have built-in support for Internet Media Types. So they can provide access to an email message's structure such as a listing of all MIME content in a message. IMAP also provides for server side searching and selection, which makes selective download of multi-part MIME content possible. These features of IMAP optimize on-line performance during a session, minimizing data transfer across the network. The next section describes MIME in more detail.

Another IMAP advantage is that the server automatically notifies the client of any state changes to a mailbox. In the case of a single user mailbox, this feature might not seem very important, but in the case of shared mailboxes this feature becomes essential. This obviously requires a continuous network connection for the server to send "unsolicited" status updates.

IMAP also has certain disadvantages, mostly in the area of resource requirements. IMAP is more complex and requires more effort to implement the server. IMAP is still not widely used, even though the original protocol was documented in 1988. IMAP4 is a functional superset of POP3; but IMAP4 is not backward compatible with POP3 or with earlier IMAP versions. As IMAP clients are not yet as popular as POP clients, this makes transition difficult. IMAP clients have to maintain the connection to the server throughout an email session, while the user reads the messages and does other

maintenance tasks on their mailboxes. But POP clients can be disconnected during this "idle" time and then reconnected to send/retrieve new messages.

POP has some disadvantages when compared with IMAP. POP3 clients usually transfer the entire contents of the mailbox, even though the *RETR* command can be used transfer individual messages. POP3 doesn't allow access to the mailbox from multiple locations. But this feature can be simulated using the "Leave Mail On Server" option provided by most POP3 clients. But messages left on the server cannot be marked as "already read;" thus a subsequent access to the mailbox will list "read" messages as "new." Some clients solve this problem by maintaining state information about read messages on the client machine. But this state information is not accessible to other computers or other clients; so the problem is once again solved only if one client on one computer is used to access the mailbox.

In summary IMAP provides a superset of POP features, which allow much more complex interactions, while making the process much more efficient; but at the cost of backward compatibility and extra resource requirements on the server side.

1.2.5 Internet Media Type

RFC822 defines the email's message body as flat text with no support for any other type of content. RFC2045 and its related documents redefines the contents of an email's message body. Internet Media Type, formerly called Multipurpose Internet Mail Extensions (MIME) is a technique for encoding arbitrary files as attachments to the Internet email messages. It allows an email's message body to be used for non-textual

messages and multi-part messages in addition to the regular textual messages already defined [RFC2045]. Internet Media Type defines the standard representation for "complex" message bodies [CONNECTED 97]. A "complex" message body can consist of binary data encoded into plain text message. A "complex" message body can also consist of multiple parts, where each part is a different binary file or a text message. In addition to this "complex" message body, Internet Media Type messages can also contain messages in languages other than English. Internet Media Type allows all this while completely maintaining backward compatibility with RFC822 message bodies.

MIME messages can be sub-divided into several parts: Message, Entity, Body part, and Body [RFC2045]. The term 'Message' either means a complete RFC822 message or a message in a message body of type "message/rfc822" or "message/partial." "The term 'Entity' refers to the MIME-defined header fields and contents of either a message or one of the parts in the body of a multipart entity" [RFC2045]. Body part refers to "an entity inside of a multipart entity" [RFC2045]. The term 'Body' means the body of an entity, which means it is either a body of a message or a body of a body part. The relationship among these parts is circular because MIME allows the possibility of one message containing another message. Figure 18 shows an example for the structure of a multi-part complex message using the Internet Media Type. In this example, the main message has four parts: plain text, binary data, message, and a mixed multipart. The third part, message, has two sub-parts: plain text and binary data. The fourth part, mixed multipart, has two subparts: an image and a message. The second sub-part of the fourth part has two parts to it: plain text and an alternative multipart. This alternative multipart has two

sub-parts: a plain text and a rich text. Using the language of [RFC2045], this example could be divided as follows: the complete example is a message and everything under "4.2" in the example is also a message; the main headers and the text are an example of an entity; each of the sections 1, 2, 3, and 4 are also entities; and sections 4.1, 4.2, 4.2.2.1, 4.2.2.2 of the example are also entities.

```
HEADER  (RFC822 header of the message)
TEXT            MULTIPART/MIXED
     1      TEXT/PLAIN
     2      APPLICATION/OCTET-STREAM
     3      MESSAGE/RFC822
            HEADER  (RFC822 header of the message)
            TEXT            (RFC822 text body of the message)
            3.1         TEXT/PLAIN
            3.2         APPLICATION/OCTET-STREAM
     4      MULTIPART/MIXED
            4.1         IMAGE/GIF
            MIME            (MIME header for the IMAGE/GIF
            4.2         MESSAGE/RFC822
            HEADER  (RFC822 header of the message)
            TEXT            (RFC822 text body of the message
            4.2.1       TEXT/PLAIN
            4.2.2       MULTIPART/ALTERNATIVE
                        4.2.2.1         TEXT/PLAIN
                        4.2.2.2         TEXT/RICHTEXT
```

Figure 18: Internet Media Type Example 1. Source: Crispin, M. "Internet Message Access Protocol - Version 4rev1." Request for Comment (RFC) 2060. http://www.internic.net/rfc/rfc2060.txt. December 1996.

To handle this "complex" message body, the Internet Media Type standard defines several new header fields in addition to the standard email headers, shown in Section 1.2.1, titled "Simple Mail Transfer Protocol." These headers can come in two locations inside a message. It is always a part of the regular RFC822 message header; in cases of

multipart messages these headers may also appear within each of the parts. The first header, "Mime-Version," identifies the MIME version used in the email. If this header is left out then it is assumed that the message body appears as it is defined in [RFC822], meaning it contains plain text message in the English language. The next two headers, "Content-Type" and "Content-Transfer-Encoding," identify the type of data and the encoding used on it. The "Content-Type" header's field data are formatted as a basic type, followed by a slash (/) and then followed by subtype. The type information is followed by additional type specific parameters. The purpose of all these is to identify the data in such a manner that the client program can deal with it appropriately by possibly displaying it, using built-in methods or with the help of an external program, or allowing the user to save it to a file. In general the top level identifier declares the general type of data and the subtype identifier specifies the data's format. In the example given in the Section 1.2.1, titled "Simple Mail Transfer Protocol," the type information identifies the email message to consist of plain text using the "us-ascii" character set. Other possible values for the "Content-Type" header is shown below with an explanation for each. Five of these general types are called "discrete types," that is types whose content is ignored as far as the MIME processing is concerned. The other two types are called "composite types," that is types that need additional handling by the MIME processor [RFC2045]. Another way of describing these two is to say that MIME, by itself, understands "composite types" and does not understand "discrete types." There are seven basic media types defined in the Internet Media Type standard and they are: *Application*, *Audio*, *Image*, *Multipart*, *Text*, *Video* [RFC2046].

The "discrete types" are described here. The *Application* media type is the generic type used for any binary data file. The most common subtype is the "octet-stream," which describes the data as uninterrupted binary data. Parameters are used for specific applications and data formats. The *Postscript* subtype is also defined for the Application media type. The *Audio* media type is used for audio files such as Sun's *au* format or Microsoft's *wav* format. The *Image* media type is used to transfer image files; the two most common subtypes are Compuserv's *gif* and Joint Picture Expert Group's *jpeg*. The *Text* media type is used for both plain text messages and formatted text messages, such as those using HTML or other conventions. Text from word processors is not considered to be part of the *Text* media type because these formats usually require an external application to view or edit them; these formats can be supplied as parameters to the *Application* media type. The character set used in the text message is specified as a parameter. The *Video* media type is used to transfer video files such as Motion Picture Expert Group's *mpeg* format. The Internet Media Type definition also specifies that all unrecognized type and subtypes should be treated as uninterrupted binary data, that is, of the type "application/octet-stream" [RFC2046].

The two "composite types" are described here. The *Multipart* type informs that an email message is composed of multiple parts. A unique boundary string is specified as the parameter in the "Content-Type" header, which is used to separate the different parts of a *Multipart* message. Several subtypes such as *mixed*, *alternative*, *digest*, *parallel*, are defined for this type, with the *mixed* subtype being the most common. The *mixed* subtype refers to a generic mix of several subtypes. The *alternative* subtypes refers to messages

containing multiple versions of the same data, perhaps in different file formats or with different level of detail. The *alternative* subtype's order has additional semantics in that the order listed in the message is from least preferred version to the most preferred version; but the client program should choose the "best" possible version of the message and only that version should be presented to the user. The *digest* subtype is meant for messages consisting a listing of many individual messages of the type "message/rfc822." This subtype is useful when presenting a compilation of several messages, perhaps those from a mailing list. The *parallel* subtypes indicates that all the different parts of the message should be presented to the user simultaneously. The *Message* media type is used to describe a message that contains other messages. The *rfc822* subtype is used to indicate that each message is a RFC 822 compatible message. The *partial* subtype is used to indicate that each part is part of a larger message that needs to be combined by the client. The *external-body* subtype is used when pointing to a message stored in another mail server or in another machine accessible by this server. The parameter is used to describe the access mechanism and the message body is used to describe the location for the actual message and its actual "Content-Type." An example of the external-body subtype is shown in Figure 19. The first "Content-Type" refers to the actual message body, which is described by the second set of headers. Other possible values for the "access-type" parameter are "ftp," "anon-ftp," "tftp," "local-file," and "mail-server" [RFC2046]. These options allows the external message to be retrieved from a file on another machine, from a file on the local machine or from a mail server on the network.

Other possible parameters for the *external-body* subtype are: expiration, size, and permission [RFC2046].

```
Content-Type: message/external-body;
              access-type = local-file;
              name = "/u/nsb/Me.jpeg"

Content-Type: image/jpeg
Content-Id: <id42@guppylake.bellcore.com>
Content-Transfer-Encoding: binary
```

Figure 19: Internet Media Type Example 2. Source: Freed, N. and Borenstein, N. "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types." Request for Comment (RFC) 2046. http://www.internic.net/rfc/rfc2046.txt. 1996.

Internet Media Type standard was designed to be transparent to existing mail protocols and systems. This is accomplished by translating binary data files into plain text messages; so in effect binary data become part of the regular plain text email message already supported by existing mail protocols and systems. There are two encoding methods used in translating Internet Media Type data to plain text messages: *base64*, and *quoted-printable*. The standard RFC822 message consists of *7bit* "us-ascii" data with each line less than 1000 characters. Most non-English language can not be represented by this 7 bit encoding; thus these languages are either encoded or sent as *8bit* message. The *7bit* messages are those that can be handled by many systems on the Internet and so this is considered to be the least common denominator. Messages sent as 8bit might not survive across platforms or mail gateways. For this reason, binary data and other languages are either encoded as *base64* or *quoted-printable*. As the name suggests, the *quoted-printable* encoding produces output that is more or less readable if the original data had not been binary. The *base64* encoding produces output that is dense

and uniform, but unreadable by humans. But *base64* is more common on the Internet because *quoted-printable* encoding increases the size of the message more than the *base64* encoding. The "Content-Encoding" header can appear either as part of the RFC822 header or in each section of a multipart message. For the header that appears at the top along with RFC822 headers, this describes the encoding used on the entire message. For the header that appears as part of a multipart message, this describes the encoding used for that part of the message.

MIME headers also allow a brief description to be attached to a MIME message by using the "Content-Description" header. Additional MIME headers can be used that are not recognized by the standard described in [RFC2045]. These additional headers have to begin with "x-" such that a new header might read "x-PriorityLevel."

1.3 Languages

Static HTML does not provide enough client side intelligence. Powerful applications on the web require a powerful user interface on the client side. Client side programming can be of two types: scripts and compiled programs. Choices such as JavaScript and VBScript exist for the script type of client side programming on the web. For compiled programs, languages such as C, C++, Delphi, Java, and Visual Basic can be used to create various types of client side programs. These client side programs are called by various names such as plug-ins, controls, or applets. This part of the chapter describes the Java programming language and the JavaScript language, in the two Sections 1.3.1 and 1.3.2. These two are chosen because the two most popular web browsers, Netscape Navigator

and Microsoft Internet Explorer, supports them; other languages and scripts are not supported by both browsers at the same level.

1.3.1 Java

A simplified way of describing Java is to say that it is "a new programming language, with elements from C, C++ and other languages, and with libraries highly tuned for the Internet environment" [LINDEN 96]. A complicated way of describing Java is to say that it is "a simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language" [SUN]. This sub-section describes Java taking a middle road between these two extremes.

1.3.1.1 Java Language Features: The Java programming language environment ("Java" for short), as the name suggests, consists of a programming language and an environment for the language's use. The language itself was designed to be object-oriented from the ground up, with strong influences from C++. But the language removes many features from C++, making Java applications more robust and easier to develop. It provides extensive compile-time and run-time checking. The memory management model has been simplified considerably, removing programmer control over pointers, which eliminates entire classes of programming errors. Automatic garbage collection, part of the Java run-time system, also increases a Java application's ease of development and reliability. Besides C/C++, Java borrows from other languages such as Eiffel, SmallTalk, Objective C, and Cedar/Mesa [GOSLING 95]. Java has elements of

concurrency from Mesa, exceptions from Modula-3, dynamic linking and automatic storage management from Lisp, interface definitions from Objective C, and ordinary statements from C/C++ [LINDEN 96]. Java language allows developers to use it for three types of programming: symbolic, numeric, and systems [KRAMER 96]. Java's object-orientedness and dynamic linking provides features similar to the symbolic programming language Smalltalk. Java's platform independent data types and well-defined arithmetic operations provides for writing stable numerical algorithms across platforms. Java's control structures and other statements similar to C and C++, allows systems programming.

Java's syntax is very similar to that of C/C++; but some of the semantics and details of other semantics have changed [GOSLING 95], [ECKEL 97], [LINDEN 96]. It has the standard primitive data types of C++, but removes the *struct* and *union* data types and adds the *boolean* data type. Java, as part of its effort to support internationalization, uses the Unicode character set for the *char* data type. Java arrays and strings are true objects, unlike in C++. Java creates two types of string objects: the *String* class is used for read-only objects and the *StringBuffer* class is for read/write objects. Java removes the *goto* statement; but provides expanded functionality using the *continue* and *break* statements. The *new* operator is used, like in C++, to allocate memory for objects; but recovering memory is left up to the automatic garbage collector which runs as a low priority thread in the Java run-time environment. Java also removes the automatic coercions feature of C++; an explicit cast is required in such situations. The biggest change from C++ to Java is the removal of all support for pointers and pointer arithmetic. Java has no free-

standing functions; instead all functions are created as methods of some class. Java also has no class declarations; only class definitions are allowed [ECKEL 97]. Java removes operator overloading, but function overloading is still available. Even though Java was built from the ground up to be object-oriented, it doesn't support multiple inheritance, only single inheritance is possible. Multiple inheritance can be simulated by using the interface mechanism, whereby a sub-class inheriting from the super-class implements the interface for another class. This feature is conceptually similar to Objective C's protocols feature [GOSLING 95]. Java also removes all support for preprocessor; the effect of the *#include* command from C/C++ is supported by using the *import* command. Constants, *#define* from C or *const* from C++, can be achieved using the *final* keyword before a variable declaration. In C++ classes, the protection keywords apply to blocks of code; in Java these keywords have to be prefixed for each member definition. Java doesn't support C++'s version of protected classes. In Java the keyword *protected* means that class member is accessible to inheritors and to others in the same package; this makes it similar to file level protection in C++. Java also doesn't support changing protection levels through inheritance. C++ *inline* keyword's functionality is replaced with a similar functionality from the *final* keyword.

An example of a Java applet is listed in Appendix E.1, showing some of the concepts described above. In this example, *Shapes* is an abstract interface listing two functions: *getArea* and *getPerimeter*. The *Coordinates* class holds the coordinates for a shapes object. The *Square* class and the *Circle* class are examples of multi-inheritance. These classes extends the *Coordinates* class and implements the *Shapes* interface. This

example also shows some of the other features of Java mentioned above, such as C++ style syntax and support for object-oriented paradigm.

Java has built-in support for multithreading at the language level with the use of the *Thread* class. Java provides synchronization primitives such as monitors and condition locks, and its system libraries are written to be thread safe. This provides a safe environment for conflict-free operation of multiple concurrent threads of execution. The *start* method and the *run* method, provided by the *Runnable* interface, are the key functions needed to make an application or an applet multi-threaded. The *Thread* class is used to create different objects for each thread of execution needed. An example of a multi-threaded applet is shown in Appendix E.2.

Java supports exception handling, which is also shown in Appendix E.2. Exceptions are unexpected events that can be caught by using the *try* statement and the *catch* statement. When any Java statement executed within the *try* block results in an error, an exception is thrown. Different *catch* blocks, containing the exception handler code, can be created to catch different exceptions. The *throw* statement can be used to throw user defined or system defined exceptions.

A Java program is organized into different levels; going from the highest to the lowest, the levels are: applications or applets consist of a set of imported packages, each of them containing one or more classes with many methods [LINDEN 96]. Java deals with these various levels with five different types of visibility and accessibility. The *public* items can be accessed from anywhere; *protected* items are accessible only within the package or the subclasses of other packages; the *default* visibility is that items can be

accessed by any class of its package; *private protected* restricts access to the class and its own subclasses; and a *private* item is only visible inside its class. Java also supports automatic garbage collection and networking [LINDEN 96].

The complete Java system includes a number of system libraries [GOSLING 95]. The Java Base System provides the minimal implementation consisting of the language package for data types, threads, exceptions and other fundamental classes; the I/O package providing streams and random-access files; the network package supporting sockets and interfaces for Internet protocols; the utility package providing container classes and other classes; and the Abstract Windowing Toolkit (AWT) which provides an abstract layer for interfacing with the graphical user interface systems. The extended set includes support for 2-dimensional and 3-dimensional graphics programming; multimedia and telephony applications; network and systems management; electronic commerce; and encryption and authentication [VONDRAK 97]. Other features such as Java Beans provide for software component building; Remote Method Invocation (RMI) allows function calls across the network. Java also provides for database access across the network and platform boundaries using the Structured Query Language (SQL) and the Java DataBase Connectivity (JDBC).

The AWT provides the ability for uniform interface design across different GUIs. It has several input objects such as *Button*, *Checkbox*, *Choice*, and *TextField* [JAMSA 96]. Java supports event driven programming like many other visual languages. The *action* method is used to define the code for a given GUI widget.

1.3.1.2 Java Environment Features: Java is a more than a programming language, it is also an environment where Java applications and applets execute. This environment is shown in the two figures of Appendix E.3. Java is designed to be a cross-platform environment that allows the creation of portable applications. Java applications are distributed as *bytecode*, also called *J-code*. *Bytecodes* are "architecture neutral intermediate format designed to transport code efficiently to multiple hardware and software platforms" [GOSLING 95]. *Bytecode* either gets interpreted by a Java interpreter or is compiled into native machine code by a Just In Time (JIT) compiler, both implemented on the target environment. Besides the architecture neutral instruction format, Java specifies the size and behavior of its data types and arithmetic operations on them. This eliminates machine dependent word size and other platform specific implementation details. There are no "implementation-dependent" notes in the Java language specification. This architecture neutral and portable language environment of Java defines the Java Virtual Machine (VM). Two different views of the Java environment are shown in Figures 53 and 54 of Appendix E.3. The Virtual Machine is implemented on top of a machine's native operating system and Java applications and applets run on top of the Virtual Machine. Java VMs are designed exactly to be the same on all systems and they are either installed as part of the web browser or as part of the operating system. The Virtual Machine insulates the Java application and applets from the differences in the underlying operating systems; thus providing an uniform programming interface on any hardware [KRAMER 96]. With this understanding, *bytecode* can be thought of as high-level machine code for the Java VM implemented by

the Java interpreter and run-time system. For an application to run on multiple platforms, the traditional scheme of binary distribution is not possible. So Java applications are compiled for a single virtual machine, which in turn must be implemented in each of the target physical machine and environment. A model of the Java platform shows how the Java Virtual Machine fits into the overall picture. The Java Platform has two main parts: the Java Virtual Machine and the two sets of APIs. The Java VM's machine dependent part is separated into the porting interface section of the platform; this makes Java VM easier to port to different hardware and software platforms. As mentioned above, the Java platform has the minimum API required for all implementations and an extended API that is optional at this time, but might move into the base standard [KRAMER 96].

The Java language environment creates a middle ground between very high level scripting languages and very low level compiled languages [GOSLING 95]. Scripting languages are portable but slow. Compiled languages are non-portable but fast. Java is both portable and provides performance that is comparable with compiled languages when used with just in time compilers. Just in time compilers improve the speed of Java; but Java is not 100% portable. Java uses the AWT to provide an abstract interface into the native window systems, such as X/Motif on UNIX, Windows on Microsoft Windows, and Toolbox on Macintosh. For Java applets and applications to be completely cross-platform, all its dependencies and libraries, including access to third part software, have to be cross-platform. The interface between the Java Virtual Machine and the native operating system or the windowing environment will always be implementation and platform dependent [SHIFFMAN 97]. Java is simpler and easier to use, like scripting

languages, and provides for powerful applications like compiled languages. Blundon's article discusses the promises made by Java and their reality [BLUNDON 96]. Java is a good environment for building client side software, but for the server side it still needs to improve. Java provides a secure environment. Even though Java applications and applets can be "written once and run anywhere," their behavior is not completely consistent across platforms. Java can increase the productivity of developers and increase the quality of the applications developed.

1.3.1.3 Java Applications: Java applications are just like any other applications, in that they have full access to the local machine that they are running on. They can read and write to the local file system and make network connections to any computer on the network. They can also invoke native code by calling externally linked functions which can be written in any language, such as C or C++. Java applications can not be run inside the browser or the applet viewer; instead the Java interpreter is used to run the Java application. Java applications also do not have any built-in downloading mechanism, unlike the Java applet which is automatically downloaded. Java applications also require the Java Platform to run; but this platform support can come in any way, including as a separate program, embedded as part of the operating system, or as part of the Java application itself [KRAMER 96]. A Java application does not require a network, unlike the applets used inside a browser; of course applets can be loaded from the local file system.

Besides the differences in restrictions, Java applications also start differently from Java applets. Since Java applications are similar to any other stand alone application and the Java language is similar to C/C++, the startup is also similar to C/C++ applications. Java applications start execution in a public function called "main" similar to the convention used in C. In fact the function signature is also similar to the "main" function in a C application. The minimum requirements for a Java application are shown in Figure 22.

```
public class JavaApplication
 {
  public static void main (String args[])
   {
    system.out.println ("The minimal Java application");
   } /* main */
 } /* JavaApplication */
```

Figure 20: Java Application Example

1.3.1.4 **Java Applets:** Java enables a programmer to extend Internet browsers by embedding Java programs, called "applets," on a web page. Applets can be thought of as small applications that can only be run from within a browser or an applet viewer. Java applets provides a developer with the flexibility to develop a more sophisticated user interface. Java applets provides the full range of event-driven pop-up windows and graphical user interface widgets [VONDRAK 97]. The main difference between Java applets and Java applications is that applets have to live inside a "sandbox" with more security restrictions. The sandbox security system is composed of several features of the Java language and the Java platform and environment operating together; these features

of the "sandbox" include: class loader, bytecode verifier, security manager, and the lack of pointers in the Java language. Other security features of the Java environment includes the Java ARchive (JAR) file format and digital signatures, which combine to prevent unauthorized modifications to the applets being used across the network. Since applets can be downloaded from anywhere and run on the local machine, security issues are very important. By default web browsers prevent all applets on the network from doing anything "dangerous," such as reading from or writing to the local file system; making connections to computers on the network other than where it came from; or executing native code on the local machine. There are two different ways that applets are loaded by a Java system: over the net or by the file system. The way an applet enters the system affects what it is allowed to do. If an applet is loaded over the net, then it is loaded by the applet class loader, and it has security restrictions enforced by the applet security manager. If an applet resides on the local file system, then it is loaded by the file system loader. Applets loaded via the file system have the same restrictions as that of Java applications. In effect the Java security model for applets creates a "sandbox" and prevents applets from reaching outside. But digitally "signed" applets from trusted sources can be allowed to step out of the "sandbox." Java applets loaded from the local file system are also allowed to step out of the "sandbox;" in fact these applets are more similar to Java applications than they are similar to Java applets loaded from across the network. Since applets are meant to be downloaded from across the network, they are usually small in size; but there are no specified limitations on the size of an applet or an application.

The Java applet execution environment is very different from the Java application execution environment. In one sense both these environments are the same, that is both of them are interpreted to run on the Java virtual machine. But Java applets execute within the browsers. In the course of a browsing session a web page containing a Java applet might be revisited or reloaded again and again. As the web page is scrolled up and down, many applets can come and go out of view. This type of behavior makes Java applets to be started, paused, resumed, stopped and restarted. In this sense an applet's environment is very different from that of the application's environment. Java provides the *init*, *start*, *run*, *paint*, *repaint*, *stop*, and *destroy* methods to handle this unique environment [JAMSA 96], [LINDEN 96]. These methods are overridden by the applet developer; but these methods are not called directly by the developer instead the run time environment automatically calls these methods as needed. The *init* method is used to initialize the applet and it is called when the applet is first loaded into memory. The *start* and *stop* methods are used to start and stop the applet, respectively. These methods are automatically called when the browser visits or leaves the page; but they can also be called explicitly by the programmer. The *run* method is not directly related to applets, but this is the function used to start a thread of execution. The *paint* method is called each time the browser needs to update the display, so this method can contain code for displaying output. This method can only be called by the browser itself; the *repaint* method is provided for situations where the programmer needs control of when the display is updated. The *destroy* method is similar to a destructor for a C++ class; this method is called when the browser discards the applet.

```
<html>
<title>Java Applet Example</title>
<body>
    <center>This page contains an applet, which is displayed
    in the panel below</center><br>
    <applet code="JavaApplet.class" width=320 height=200
            align=middle vspace=25
            alt="This browser doesn't support Java Applets
                 or it is disabled."
            param name="Title" value="Java Applet Example">
    </applet>
</body>
</html>
```

Figure 21: Java Applet Example Part 1 of 2

Java applets can either be run from within the applet viewer or they can be run from within the browser. Figure 21 shows the HTML code needed to run an applet within the browser. This web page displays a message and loads an applet below that message. The Java code for the applet is shown in Figure 22. The applet code retrieves the parameter specified in the HTML by calling the *getParameter* method when the applet is initialized. The paint method is used to display the parameter and some environment variables from the Java environment on the client side.

```
import java.awt.*;
import java.applet.*;

public class JavaApplet extends Applet
 { String AppletTitle;
   public void init()
    { AppletTitle = getParameter ("Title");
      if (AppletTitle == null)
        AppletTitle = "Not Specified"
    } /* init */
   public void paint (Graphics g)
    { g.drawString ("Java Applet's Title is " + AppletTitle,
                    5, 15)
    } /* paint */
 } /* JavaApplet */
```

Figure 22: Java Applet Example Part 2 of 2

1.3.2 JavaScript

"The intersection of Java and JavaScript is the empty set" [LINDEN 96]; that is, Java and JavaScript have very little to do with each other. JavaScript is a simple scripting language with a similar syntax to that of Java; but it has very little functional similarities with Java. JavaScript provides the power of control structures and conditional tests with the simplicity of a scripting language. JavaScript can be used both on the client side and on the server side. On the client side, JavaScript makes it easy to enhance web pages and validate form input. On the server side, JavaScript makes it easy to access databases from any vendor on any platform. JavaScript is a compact, object-based scripting language for client-side and server-side "programming."

JavaScript's syntax is similar to Java's syntax; but it doesn't support many features of Java, such as static binding and strong type checking. JavaScript objects can be dynamically created, without declaring them beforehand, and they are automatically created when needed. JavaScript supports four basic data types: a generic object type, a number type that can hold integers or floating point number, a string type and a boolean type. It also has arrays and pointers to functions. Most of the regular control structures and loop structures are supported. JavaScript is object-based rather than object-oriented; this means lack of support for classes and inheritance.

JavaScript is primarily intended for interaction with HTML pages and the web browser itself; for this purposes it exposes many tags and controls of the web page and its browser. An *object.variable* notation can be used to get and set many properties of the web page and the web browser. This allows JavaScript scripts to modify HTML pages,

while Java applets usually run inside a sub-area of an HTML page with very little control outside this sub-area. Some of the objects exposed by JavaScript are shown in Table 1.

Table 1: JavaScript Objects. Source: Shah, Rawn. "Beginner's JavaScript." http://www.javaworld.com/javaworld/jw-03-1996/jw-03-javascript.intro.html. JavaWorld. 1996.

Objects	**Description**
Anchor	HTML anchor tag object
Applet	Java applet object
Button, Checkbox, Password, RadioButton, Reset, Submit, Text, TextArea	Objects for different items on an HTML form
Date	Date object and methods to manipulate and display it.
Document	Object describing the HTML page
Form	An object for the entire HTML form
History	History list of the browser
Link	HTML link object
Location	Object describing a URL
Math	Mathematic constants and functions
Selection	A text selection inside an HTML form's textarea item or an text input item.
String	A generic string object with methods for manipulation and display
Window	Object describing the current browser window or a document window

JavaScript "programming" is an event-driven programming model. JavaScript scripts usually modify the web page in place or they interact in response to events generated by the user. New HTML tags are used for the purposes of JavaScript. Specifically the *<script>* tag is used to contain the JavaScript scripts. The *<script>* section of the HTML page can be inside the *<head>* section, inside the *<body>* section, or both. JavaScript script placed inside the *<head>* section is executed before the rest of

the HTML page is displayed. JavaScript also adds several parameters to HTML tags. These parameters are used to call JavaScript functions in response to events. These functions can be used for error checking or other types of data validation before the user input data is sent to the server. They can also be used to control what the browser displays.

Several event handlers are defined for the objects of an HTML form [SHAH 96]. These event handlers are: *onBlur*, *onChange*, *onClick*, *onFocus*, *onLoad*, *OnMouseOver*, *onSelect*, *onSubmit*, and *onUnload*. The *onBlur* and *onChange* events are fired when an HTML form item looses focus or the value is changed, respectively. The *onFocus* is fired when items get focus. The *onClick* event is fired when HTML form items such as *Button*, *Checkbox*, or *RadioButton* is clicked on or when selected. This event is also fired when an HTML link is clicked on. The *onLoad* and *OnUnload* events are fired when an HTML window or a frame is loaded and unloaded, respectively. The *onMouseOver* event is fired when a mouse cursor moves over a link on the HTML page. The *onSelect* event is fired when some text inside the HTML form's text field or textarea field is selected.

```
<html>
<head>
    <title>JavaScript Example</title>
    <script language="JavaScript">
    <!--
        document.write ("This text will be displayed before
        the actual body of the HTML page is displayed.");
        document.write ("Current Date & Time is " + Date);
    -->
    </script>
</head>
<body>
    <script language="JavaScript" src="jscript.js"></script>
    <H1>
        An example of JavaScript script that validates a
        HTML form's data
    </H1>
    <br>
    <form>
        <input type="checkbox" name="OS_Type"
               value="Win3.x/95" onchange="set_os(this)">
        <input type="checkbox" name="OS_Type"
               value="Mac OS8" onchange="set_os(this)">
        <input type="checkbox" name="OS_Type" value="Unix"
               onchange="set_os(this)">
        <input type="checkbox" name="CPU_Type" value="x86"
               onchange="set_cpu(this)">
        <input type="checkbox" name="CPU_Type"
               value="68000s/PowerPC"
               onchange="set_cpu(this)">
        <br>
        <input type="button" value="OK"
               onclick="check_matching(this.form)">
    </form>
</body>
</html>
```

Figure 23: JavaScript Example Part 1 of 2

An example of a JavaScript script is shown in Figure 23. This script displays the current date and time in the *<head>* section of the web page. It also validates the data from an HTML form and displays an error message for invalid data. As the example shows, it is common practice to enclose the script inside a comment to prevent older browsers from misinterpreting this code. In this example, the script code in the *<head>*

section is embedded along with the HTML code. As an alternative the "SRC" parameter can be used to store the JavaScript code in a separate file; as shown for the script in the *<body>* section. The script from that separate file is shown in Figure 24.

```
var OS_value;
var CPU_value;

function set_os (OS_Type)
  {
   OS_value = OS_Type.value;
  } /* set_os */

function set_cpu (CPU_Type)
  {
   CPU_value = CPU_Type.value;
  } /* set_cpu */

function check_matching (inputform)
  {
    if (OS_value == "Win3.1/95") AND (CPU_value != "x86")
      {
        document.write ("Win3.1 or Win95 can only run on x86
                         platform.");
      } /* if */
    if (OS_value == "Mac OS8") AND
       (CPU_value != "68000s/PowerPC")
      {
        document.write ("Mac OS8 can only run on 68000s
                         or PowerPC platform.");
      } /* if */
  } /* check_matching */
```

Figure 24: JavaScript Example Part 2 of 2

A comparison of Java and JavaScript is presented in Table 2. For Java, only the applets are considered since Java applications can not run inside a web browser. Bruce Eckel says that 80% of client-side programming problems can be solved using JavaScript; but Java is needed for the remaining 20%, the "really hard stuff", which can not be solved using scripting alone [ECKEL 97].

Table 2: Comparison of JavaScript and Java

JavaScript	**Java**
Interpreted by the web browser.	Compiled into bytecode and then interpreted by the web browser and the Java Virtual Machine.
Object-based: Built-in objects and their methods and properties can be used, but classes and inheritance are not supported.	Object-oriented: Classes with single inheritance is supported, in addition to the built-in objects; multiple inheritance is possible using the interfaces mechanism.
JavaScript code is embedded and integrated with HTML code; but it can be separated into a different file.	Java code is separate from HTML code; but Java applets are accessed from HTML pages.
Loose typing and Dynamic binding.	Strong typing and Static binding.

CHAPTER 2

REQUIREMENTS ELICITATION

Requirements Elicitation is one of the earlier stages of Requirements Engineering (RE), which itself is one of the earlier stages of software development process. Requirements Engineering in general consists of detecting the requirements, validating and verifying them for correctness and consistency, and representing them in an understandable manner. The three main goals of the RE process are to improve the understanding about the system and transform this understanding from an informal representation into a formal representation, while achieving a common agreement among all the people involved. These three goals of the RE process are represented as the three dimensions of Requirements Engineering, specification, agreement, and representation in [POHL 93]. This leads to the four general processes of RE: requirements elicitation, requirements analysis, requirements validation, and requirements specification [RAGHAVAN 94]. Requirements Elicitation is defined as "the process through which customers, buyers, or users of a software system discover, reveal, articulate, and understand their requirements" [RAGHAVAN 94]. Requirements Elicitation can also be thought of as the process of gathering different viewpoints from different sources and reaching an agreement on a common viewpoint. "The prime objective of the requirements definition process is to achieve agreement on what is to be produced" [BRACKETT 90]. The Requirements Elicitation process has also been called identifying, gathering, determining,

formulating, extracting, or exposing requirements [RAGHAVAN 94]. Each of these different terms expresses the different connotations that exist in the process.

[RAGHAVAN 94] describes the Requirements Elicitation steps as:

- Identify the sources of requirements,
- Gather information about their needs,
- Analyze the information for implications, inconsistencies, or unresolved issues,
- Reconcile the differences in understanding between users and analysts, and
- Generate the requirements statements.

These steps are typically iterated until a complete and common understanding of the system to be developed has been reached among the people involved. The people involved in the Requirements Elicitation process are: analysts, developers, customers, and users.

Requirements Engineering is a very important step for a project's success [RAGHAVAN 94], [POHL 93], [DAVIS 93]. It has been shown that "the later in the development life cycle that a software error is detected, the more expensive it will be to repair" and that many errors remain undetected until development has "progressed" beyond the stage in which the errors were made [DAVIS 93]. It has also been shown that it takes more time to fix errors in later stages of development as compared to earlier stages of development. Another observation is that errors made in earlier stages build up and cause more errors in later stages. Experience has shown that incorrect, incomplete, or misunderstood requirements are the most common causes for poor quality, cost overruns,

and late delivery of software systems [RAGHAVAN 94]. Requirements Elicitation's importance is directly tied to the significance given to Requirements Engineering in general because eliciting requirements is the first step of Requirements Engineering. So the Requirements Elicitation process is essential for the development of quality software products [MILLER 93], [CHRISTEL 92].

Because of the importance given to Requirements Elicitation, many techniques have been developed for this process. Many techniques have elaborated on the general procedure described above. Some are high-level frameworks, process models, or methodologies that provide general guidelines for eliciting requirements. Others are low-level techniques or methods that provide specific tactics for eliciting requirements. These techniques give detailed processes, specific questions or categories of questions to ask, structured meeting formats, individual or group behaviors, and templates for organizing and recording information [RAGHAVAN 94].

2.1 Requirements Elicitation Framework

Existing models, methodologies, and techniques for Requirements Elicitation had failed to adequately address the problems inherent in the Requirements Elicitation activity [CHRISTEL 92]. The problems not addressed by existing methods were that of scope, communication, and volatility. The problems of scope are those dealing with ill-defined system boundaries and unnecessary design information [CHRISTEL 92]. Communications problems in the Requirements Elicitation process were of two types: understanding among the participants of the process and understanding among those

affected by the process [MILLER 93]. The problem of volatility is the changing and evolving nature of requirements [CHRISTEL 92]. In response to this, the Software Engineering Institute (SEI) took the existing processes, methodologies and techniques of Requirements Elicitation and combined them to form a framework, which presents them in relation to each other.

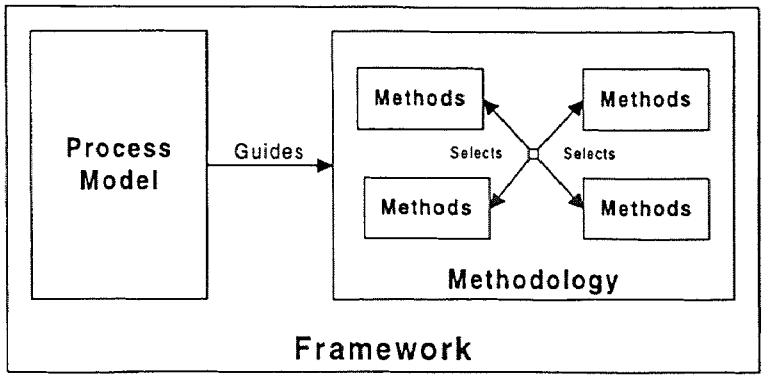


Figure 25: Requirements Elicitation Framework

SEI's Requirements Elicitation framework consists of a process model, a methodology, and a set of techniques or methods. Figure 25 shows this framework and its elements in relation to each other. The idea behind the framework is take individual methods and techniques and combine them into a methodology which can be tailored for each situation [CHRISTEL 92]. For this purpose the process model is used to guide the methodology. The process model provides a strategy that improves upon the problems of Requirements Elicitation mentioned above. The methodology recommends the methods and techniques to be used based on the situation at hand [MILLER 93].

The rest of the chapter describes this framework in more detail. The next section of this chapter describes the process model for the framework. Methodology and Methods

along with the different stages of the process model and the activities of each stage in the process model are then discussed in the last section of this chapter.

2.2 Requirements Elicitation Process Model

Requirements Elicitation deals with fact-finding, information gathering, and integration [CHRISTEL 92]. Later on, the gathered information has to be validated. Elicitation implies communication between different sets of people: analysts, customers, developers, and users. The requirements analysts are responsible for the capture of system requirements from the user community and its communication to the developer community [CHRISTEL 92]. The analyst is the middleman between the user and the developer. The analyst also has to make sure other people, such as customers, are involved in the Requirements Elicitation and that all the affected groups have a common understanding of these requirements. Recognizing the importance of this communication, a structured model for this process of communication is created as the main part of the framework. This process model governing the framework is shown in Figure 26.

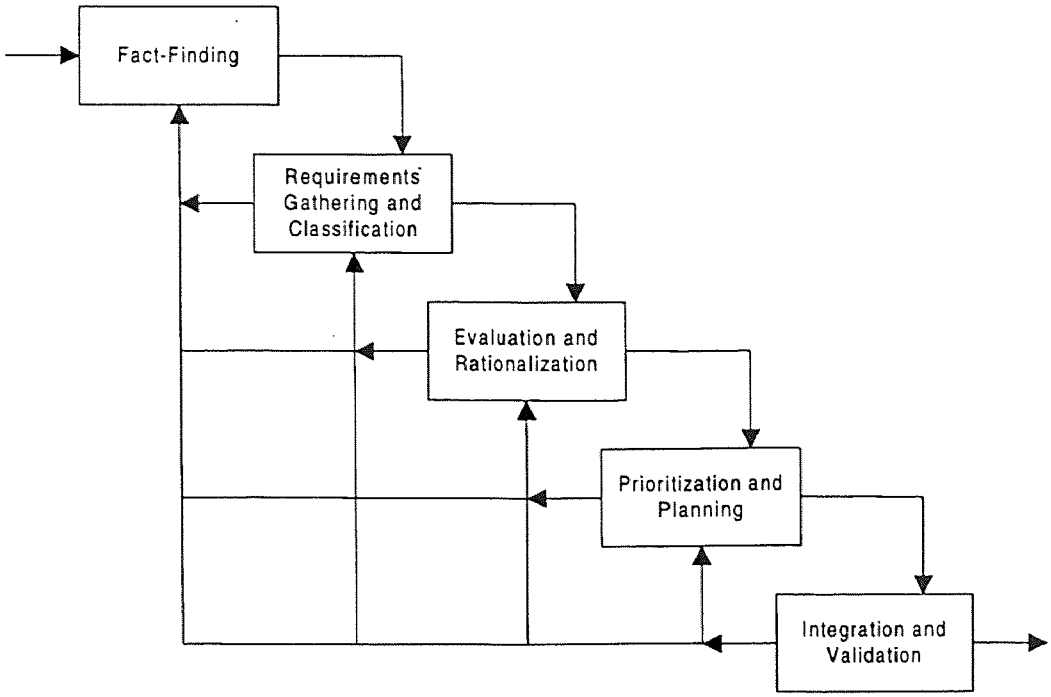


Figure 26: Requirements Elicitation Process Model. Source: Christel, Michael G. and Kang, Kyo C. *Issues in Requirement Elicitation*. Technical Report CMU/SEI-02-TR-12 or ESC-TR-92-012. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. September 1992.

In this process model each stage provides a feedback loop to the previous stages, showing that Requirements Elicitation is iterative and not necessarily linear. Iteration is necessary for reducing the complexity of the Requirements Elicitation and to handle the changing requirements. The iterative nature of the process model allows the requirements to be created in chunks and incrementally [MILLER 93]. The different stages of the process model combine the available methods and techniques into one model. Within the unified model, the stages are separated to achieve certain objectives that are necessary for the creation of the final set of requirements. Each stage has separate sets of activities that are meant for users of the system and its developers, user-oriented tasks and developer-

oriented tasks, respectively. This is similar to the notion of C-requirements, customer/end-user requirements, and D-requirements, developer requirements, mentioned in [BRACKETT 90]. The user-oriented tasks and the developer-oriented tasks for each of the stages in the process model are shown in Table 3. Section 2.3 discusses these tasks in more detail, for each of the stages.

Table 3: Requirements Elicitation Process Model's Tasks. Source: Christel, Michael G. and Kang, Kyo C. *Issues in Requirement Elicitation*. Technical Report CMU/SEI-02-TR-12 or ESC-TR-92-012. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. September 1992.

Stages	User-Oriented Tasks	Developer-Oriented Tasks
Fact-Finding	Identify relevant parties.	Identify domain experts.
	Determine operational and problem context.	Identify domain and architectural models.
	Identify similar systems.	Conduct technological surveys.
	Perform context analysis.	Assess cost/implementation constraints.
Requirements Gathering and Classification	Get wish list.	Classify wish lists.
Rationalization and Evaluation	Perform abstraction to answer questions of the form "Why do you need X?"; this in effect moves from statements of "how" to statements of "what."	Perform risk assessment.
	Capture rationale to support future requirements evolution.	
Prioritization and Planning	Determine criticality.	Prioritize requirements based on cost and dependency.
Integration and Validation	Address completeness issue	Resolve conflicts
	Check that requirements are in agreement with the original goals	
	Obtain authorization to move to the next step of development	

Both sets of these tasks are important. They complement each other: the user-oriented tasks study the user community and the developer-oriented tasks study the technology.

> Studying user needs is a first step to any solution, along with gaining an understanding of available technologies and existing tools. These two tasks interact. Without an understanding of technologies one may aim for the impossible, and without an understanding of needs, one may solve the wrong problem [Quoted in CHRISTEL 92].

2.3 Requirements Elicitation Methodology and its Methods

The difference between a methodology and a method is that methodology is more general. A methodology "offers a set of guidelines or principles which in any specific instance can be tailored" [Quoted in CHRISTEL 92] for each situation; but a method describes a single procedure for all situations. A method is described as "consisting of a grammar of steps and principles for applying them rather than just a collection of notations" [CHRISTEL 92].

The methodology described in [CHRISTEL 92] consists of several sets of activities described in five stages: fact-finding, requirements gathering, evaluation and rationalization, prioritization and planning, and integration and validation. The stages are executed according to the process model described above [MILLER 93]. The stages are briefly described here and described in detail, along with their activities, in the sub-sections.

In the fact-finding stage, the client organization is examined and facts are collected regarding high level goals of the system to be implemented. Existing similar systems are

identified and high-level constraints are also determined. The second stage, requirements gathering, captures information which determines what is to be built. The Evaluation and rationalization stage identifies inconsistencies in the gathered information and also describes the reason for the gathered information to be expressed as a requirement. The next stage, prioritization and planning, orders the requirements according to their relative importance and plans for addressing them in that order. The final stage, integration and validation, combines all the information gathered in the previous stages and creates a set of requirements. These requirements are validated to determine if they meet the original high-level goals of the client, gathered in the first stage.

As mentioned above, elicitation implies a communication loop between two sets of people. Section 2.2 described a process model for this communication loop. In each stage of the process model, a set of activities is described here for both sets of people. One set of activities is for the user community and the other set is for the developer community.

Besides the methods to be discussed below, there are other alternatives for Requirements Elicitation: determining directly, deriving from existing systems, normative analysis, strategy set transformation, critical success factors, key indicator analysis, prototyping, scenarios, and information needs analysis [CHRISTEL 92].

2.3.1 Fact-Finding

This is the first phase of the Requirements Elicitation process model discussed above. In this phase the main goals are "determine what problem is to be addressed, who needs to

be involved in the decision making process, and who will be affected by the problem's formulation and solution" [MILLER 93]. More specifically these goals are separated into tasks and they are distributed between users/customers and developers. These tasks are: identify relevant parties; identify domain experts; determine operational and problem context; identify domain and architectural models; identify similar systems; conduct technological surveys; perform context analysis; and assess cost/implementation constraints [CHRISTEL 92]. The output of these activities are "defined to be a statement of problem context, a statement of the overall objectives, and supporting representations of the boundaries and interfaces of the system" [MILLER 93].

For well-understood problem domains, the execution of this phase need not be as complex as in other cases [CHRISTEL 92], [MILLER 93]. Since a general understanding may already exist in these problem domains, a single iteration of this phase may suffice. If such an understanding does not exist, multiple iterations may be needed, looping back from the validation phase. Even if multiple passes are needed, later passes through this phase need not be as complex as earlier phases [CHRISTEL 92].

The Joint Application Design (JAD) technique; structured interviews; graphical Issue-Based Information System (gIBIS); Organizational Requirements Design for Information Technology (ORDIT); Customers Actors, Transformation process, Weltanschauung (world view) Owner, and Environmental constraints (CATWOE), objectives analysis model, domain models and technical surveys are some of the techniques that are used in this phase of the process model [MILLER 93].

JAD is used as a framework for the activities of this phase. A tailored version of the JAD technique, as used in this process model, consists of several stages executed in order: project research, preparation, session, and the final phase [MILLER 93]. In the project research stage, structured interviews are used to capture information which can be documented using gIBIS. In the preparation stage, ORDIT, CATWOE, objectives analysis models, domain models, and technical surveys are used to represent the acquired information. The formalized representation is then checked for conflicts, inconsistencies, and unresolved or missing information. The JAD sessions are used to gather the missing information and correct other errors in the gathered information. In the final phase, newly acquired information is integrated with previously gathered information. Once again the information is checked for conflicts, consistency and completeness. If any issues remain after this check, the fact-finding phase is reiterated, otherwise this phase is considered to be finished.

2.3.2 Gathering and Classification

In the phase the main goal "is to obtain information regarding what is to be built in relation to the goals, objectives, and constraints developed in the fact-finding stage" [MILLER 93]. This stage lists two main activities: get wish list for the users/customers and classify wish lists for the developers [CHRISTEL 92]. Wish lists have to be gathered from users and customers, which are then classified into different viewpoints by the developers. The output of these activities are "representations and documents detailing the customer and user oriented objectives and needs" [MILLER 93].

JAD is used as the framework for this phase also. Structured interviews and questionnaires are used to capture the information directly from the end-users and other stakeholders. The gathered information's underlying rationale is then obtained using gIBIS. Many users and customers providing requirements information, will generate many different points of view for the system to be built. Multiple viewpoints make it difficult to analyze and identify conflicting view points and other inconsistencies. COntrolled Requirement Expression (CORE) method can be used to organize these viewpoints. Viewpoints are further divided into meaningful components to handle the problem of changing requirements and incremental requirements development. Entity diagrams and data flow diagrams can be used to model these components [CHRISTEL 92].

In this phase, the JAD technique starts with the problem research stage which gathers the objectives, needs, and requirements from the users and customers. Structured interviews, questionnaires, observations, and Scenario Based Requirements Elicitation (SBRE) are used to gather information during this stage [CHRISTEL 92], [MILLER 93]. The JAD preparation stage organizes and evaluates information obtained in the previous step. The JAD session discusses and compares the requirements gathered in this phase and their relation to the objectives, goals, and constraints gathered from the fact-finding phase. Any conflicts found in the preparation stage are discussed during the session. The last stage of the JAD technique, the final phase, formally documents the information from this stage of the process model. This documentation also includes any conflicts found in this stage.

2.3.3 Evaluation and Rationalization

"The goal of this phase is to fully develop and evaluate the underlying rationale behind the requirements gathered to this point" [MILLER 93]. The activities for this phase are: to rationalize about the requirements for the users/customers and to perform risk assessment for the developers [CHRISTEL 92]. The purpose of these activities is to ensure completeness and consistency of the requirements gathered. The objectives, goals, and constraints developed in the first phase of the Requirements Elicitation process model are compared with the requirements detailed in the second phase of the process model. The comparison is performed to see if the requirements address the right issues meeting the right goals and solving the right problems. A series of interviews, between the analyst and the stakeholders, is needed to evaluate the requirements model against the rationale provided by the usage of gIBIS in earlier phases. This evaluation identifies missing rationale and unnecessary items. This rationalization process also identifies true requirements that are hidden behind the rationale [CHRISTEL 92]. Technical surveys from the first phase of the model are used to perform risk assessment in this phase.

The gIBIS method is used for capturing the rationale behind the requirements; its issues-positions-arguments framework is well-suited for this purpose. Domain analysis and its models, such as features model and entity-relationship model, are also very useful in this phase. Domain analysis is the "definition of features and capabilities common to systems in advance of software development" [CHRISTEL 92]. The entity relationship model is useful for communicating to the developers the issues for end-users, and the features model is useful for communicating to the end-users the issues for the developers.

2.3.4 Prioritization and Planning

"The goal of the prioritization phase is to arrange the requirements in order of relative importance from the view of the client and view of the developer" [MILLER 93]. The activities of this phase consist of a review of the requirements and arrangement of them based on mission criticality, cost, dependency, user needs and ability of the requirements to be incremented [MILLER 93], [CHRISTEL 92]. The Quality Function Deployment (QFD) method is used to prioritize the requirements gathered in the earlier phases of the process model.

Once again, a tailored version of the JAD is used as the framework for this phase. In this phase the JAD process starts with the preparation stage in which the QFD inputs are arranged and organized for the participants of the meeting. QFD inputs come from the requirement models, objectives, goals and constrains created in the earlier phases of the process model. The next phase of the JAD process, the JAD session, is used to construct the QFD matrix from its inputs. The matrix is based on the "wants" of the user community and the "hows" of the developer community. "The 'wants' and 'hows' create the two community's desires and abilities" [MILLER 93]. In the final phase of the JAD process, the completed QFD matrix is evaluated and reviewed by the stakeholders to finish the prioritization phase of the process model.

2.3.5 Integration and Validation

The goal of this phase is to "reduce the conflicts found in the requirements, to address completeness and to validate the requirements" [MILLER 93]. Activities in this phase

consist of checking for completeness by filling in uncompleted requirements and consistency, conflict, and validation checking to determine if the requirements meet the original goals, objectives, and constraints from the fact-finding phase of the process model. Outputs of this phase are a set of requirements; if they are complete then the Requirements Elicitation process is considered to be finished, but if they are incomplete then more iterations through the process model are needed to complete the requirements.

It is important for stakeholders to be involved in this phase of the process model. If the final integration and validation is performed by the developer community it could be viewed as the developers' interpretation of the requirements. A sense of shared ownership among the developer community and the user community would be lost in such an event.

The JAD technique is used as the framework for this phase of the process model. The primary contribution of the JAD technique is in its use as "a means to validate information already gathered" in earlier phases [CHRISTEL 92]. The JAD process starts with a preparation phase in which the analysts organize and package all the documents and models from the previous phases of the process model. The analysts review these documents and models for consistency, completeness, and validity. The JAD session is used to resolve any open issues and the priority of the requirements are reviewed. A decision on how to proceed is determined based on this review of requirements. The decision is either to proceed to the next step in the software development or to remain in the Requirements Elicitation phase and reiterate through the process model.

CHAPTER 3

REQUIREMENTS ELICITATION WITH INTERNET TECHNOLOGIES

3.1 Introduction

Requirements Engineering is a very important step for a software development project's success. Requirements Elicitation, considered to be the first step in Requirements Engineering, is the focus of this thesis. Requirements Elicitation is mainly about communications among different people within and across companies. During this part of Requirements Engineering, various stakeholders need to be able to communicate their requirements to the developers, and the developers need to be able communicate their understanding and generate feed back to the stakeholders for validation. The problem is that Requirements Elicitation techniques currently used produce a lot of documentation and require a lot of communication between and among stakeholders of a product and the developers of the product. Easing communications between stakeholders and developers makes the process of eliciting requirements easier, leading to better requirements specification. Better specification, ultimately, leads to a better product. The hypertext metaphor of the World Wide Web (WWW) on the Internet is used as a vehicle for easing communications between stakeholders and developers. It is hoped that applying Internet technologies will ease the process for Requirements Elicitation.

Several approaches for automated Requirements Elicitation exist; the approaches taken are computer-assisted group processes, automated analysis of documents, automated requirement verification, and CASE prototyping [PLAYLE 96]. This chapter

describes the application of Internet technologies for the problem of Requirements Elicitation using SEI's framework for Requirements Elicitation. In this chapter, the Requirements Elicitation Process through Internet (REPI) is described in detail. REPI is a prototype web application developed to implement the ideas of using the SEI framework to elicit requirements through the Internet. REPI is a groupware type system that uses the Internet as the meeting place for the group to meet for the purposes of eliciting requirements. Many Internet technologies could be used to develop a web site. Each technology has its own learning curve and its own benefits and limitations. The World Wide Web (WWW) technology of the Internet is used as the platform for this system. Specifically, the REPI web site uses the technologies of HTTP, HTML, Style Sheets, and JavaScript, as described in Chapter 1, to build a demo product for eliciting requirements using SEI's framework for Requirements Elicitation described in Chapter 2.

This chapter of the thesis describes the REPI demo web site and evaluates its design, in Sections 3.2 and 3.3 respectively. Section 3.2 is sub-divided along the major parts of the web site: SEI's Requirements Elicitation process model parts of the web site and the utility parts of the web site. Section 3.3 uses the framework for web site design evaluation described in [HONG 97] to evaluate the REPI web site's design.

3.2 REPI Web Site Description

This section of the thesis describes the REPI web site which is to be used as the platform for eliciting requirements in a distributed and an asynchronous manner. It is distributed because members of a given project need not be in the same location, they need not meet

physically. They contribute their information electronically using the Internet and the REPI web site. Requirements and other information are collected and stored in a database on the server which is connected to the web site. As different users from different locations log in, the collected knowledge of the whole project is displayed to each and every member of the project. It is asynchronous because users need not be logged in at the same time and these people are not only separated by distance; they could also be separated by time zones. Individual users are working alone, at their own pace and at the time of their own choice. But the whole project's work is collected and the most up to date information is displayed to all the users at the same time.

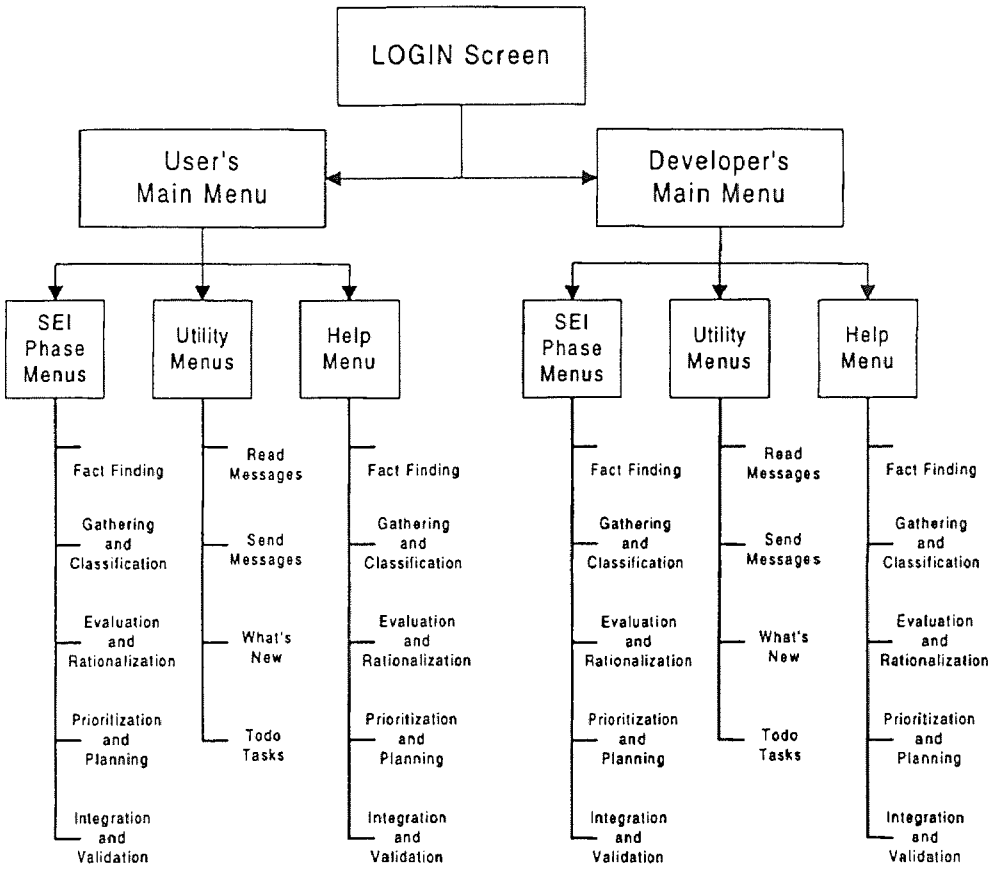


Figure 27: REPI Web Site Structure Overview

The REPI web site is organized as a series of pages branching from the two primary menu pages, as shown in Figures 27 through 29. The two primary menu pages divide the complete set of web site users, for a given project, along their lines of responsibility: the client side people are responsible for providing the requirements for a product, and the development side people are responsible for understanding the product. Each of the two primary menus branches of into another set of menus, one for each of the five phases of the SEI's Requirements Elicitation process model. Figures 28 and 29 show the web site structure for the client side users and for the development side users, respectively. Each menu of this set, for each of the five phases, displays a different web page based on the tasks of that phase. Besides the pages from this main structure, several additional menus and pages are attached as common items to the two primary menu pages and to all the other pages branching off them. A "Login" page is included as the main page for the REPI web site, this page is used to branch off into the two primary menu pages. The "Login Screen" is described in Section 3.2.1. The web pages branching off from the primary menu structures are described in Sections 3.2.2 to 3.2.8. The additional menus and pages are described in Section 3.2.9.

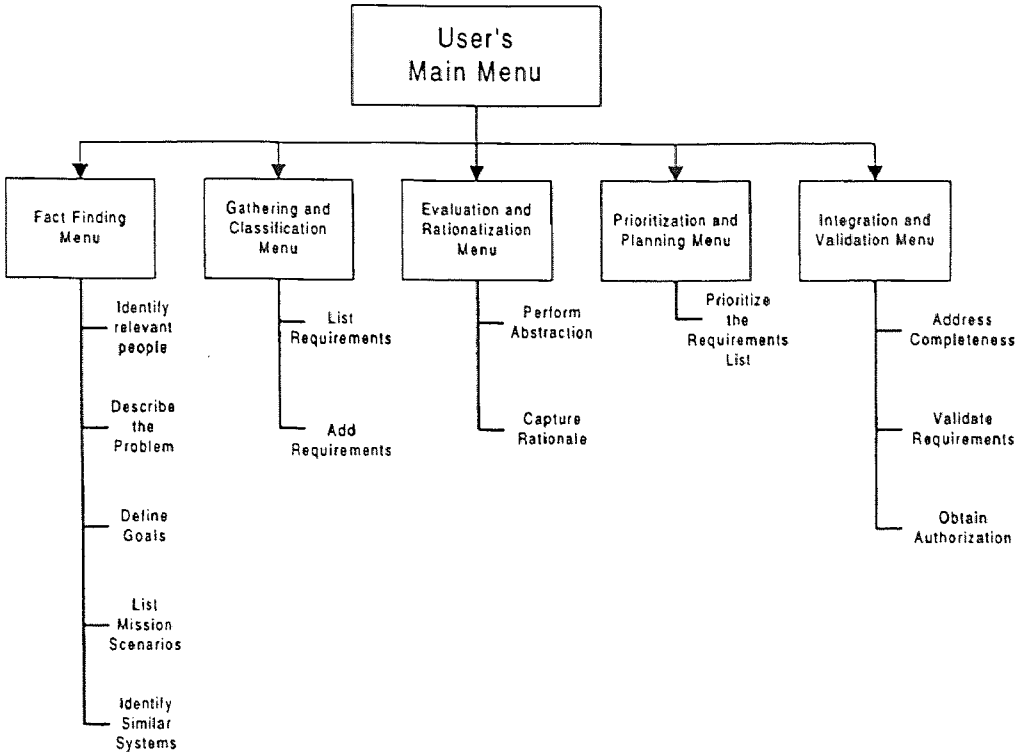


Figure 28: REPI Web Site Client Side Structure

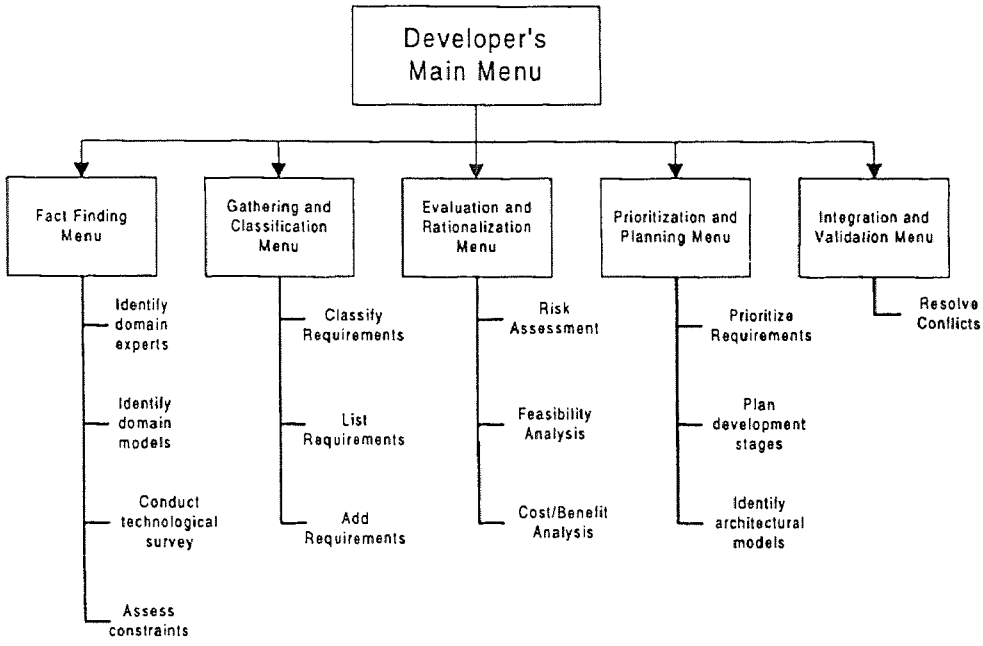


Figure 29: REPI Web Site Developer Side Structure

The screen shots shown in this chapter are from the Netscape Navigator Ver. 4.03 browser as displayed on the Microsoft Windows 95 platform. The appendices of this thesis lists the source code and shows the screen shots for the selected web pages of the REPI web site. For the sake of print quality, the background images in the bottom two frames of the REPI web site's pages are not printed and for the sake of the thesis' size, some of the pages are compressed and shown along with other related pages. The appendices also shows several screen shots from various other browsers and platforms. As the REPI web site developed for this thesis is a demo version, none of the back-end functionality described are implemented at this time. The demo version displays an error message whenever these parts of the web site are accessed.

3.2.1 Login Screen

[AL-RAWAS 96] describes the problem of requirements traceability as the inability to trace the human source for the actual requirements and their related information. Requirements traceability becomes very important during later stages of the Requirements Engineering, for validation and review. Traceability is also important during later stages of the software development cycle, if changes need to be made to the requirements or if more detailed information is necessary later on. [AL-RAWAS 96] shows that most requirements are only linked to people by their job titles, user groups or departments. In long term projects, people could be prompted, change groups or even companies. Requirements linked to an individual's name serve traceability better than other forms of linkage.

The REPI web site could be located either on an internal server within the companies Intranet or on a server on the Extranet, shared by many cooperating companies. It could also be on a public server connected to the Internet, in cases where requirements are elicited from the general public or from people who work for many different organizations. All this connectivity requires some level of security.

For all these purposes, REPI requires a unique method of identification for all project members using the REPI web site. A user id and a password are needed before any person is allowed to use the REPI web site; for this a "Login Screen" is created as the home page for the REPI web site. Figure 30 shows this home page as it is displayed by Netscape Navigator Ver. 4.03 on the Microsoft Windows 95 platform. Appendix F.1 shows the "Login Screen" as it is displayed on different platforms by various web browsers.

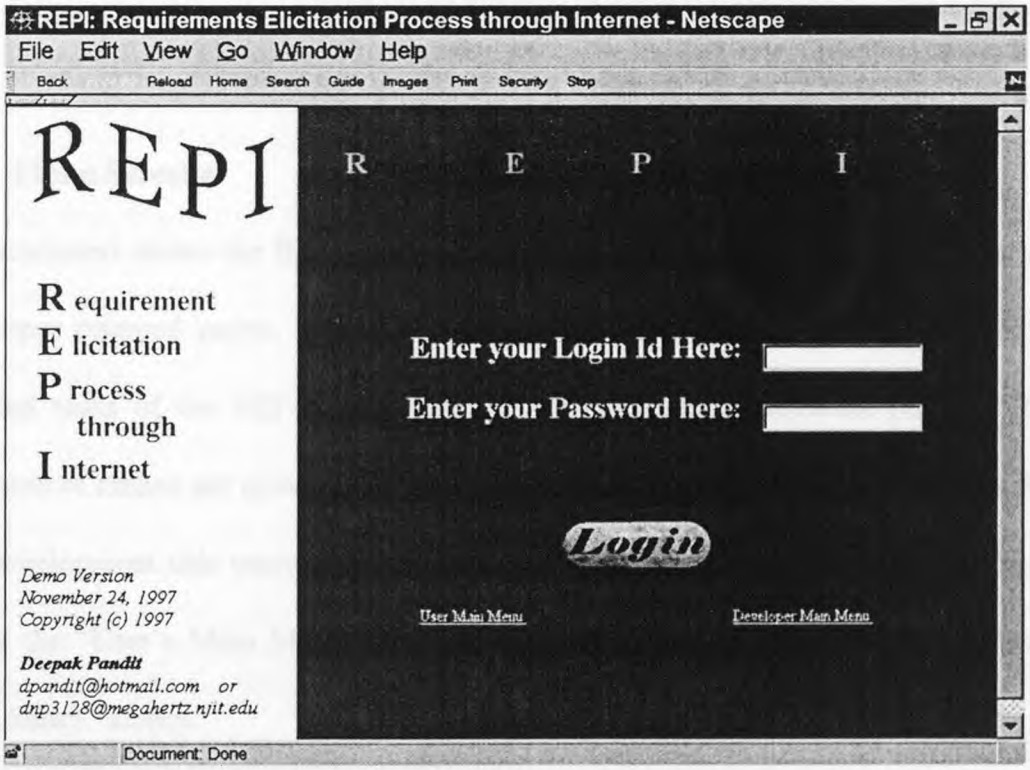


Figure 30: "Login Screen" of the REPI Web Site

This is the main page for the REPI web site and it provides a simple user id and password based security for the REPI web site. Initially project managers from the client side organization and the development side organization will be provided with a management level user id and password. These managers will identify other people who should be involved in the project using the "Identify Potential Stakeholders" task and the "Identify Domain Experts" task of the Fact Finding Phase. As people are identified, they will be given either user level or management level access as required. They will also be partitioned as either client side users or development side users. The back end process for this page should either load the "User's Main Menu" or the "Developer's Main Menu" based on the type of login id and password entered by the user. As the back end process is not implemented in the demo version of the REPI web site, two direct links are provided for these two parts of the web site; clicking on the "Login" button displays an error message.

3.2.2 Menu Screens

As mentioned above the REPI web site is divided between user oriented pages and the developer oriented pages, following the division of user oriented tasks and developer oriented tasks of the SEI's framework for Requirements Elicitation [CHRISTEL 92]. Two sets of menus are provided for these two types of project members, client side users and development side users. Figure 31 shows the two menu screens. Part 1 of Figure 31 shows the "User's Main Menu" screen and part 2 of the figure shows the "Developer's Main Menu" screen.

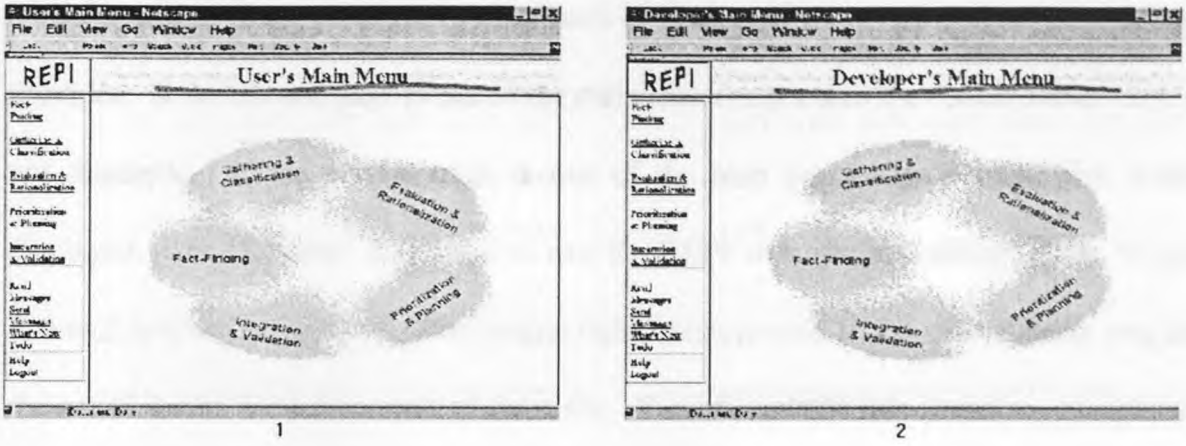


Figure 31: REPI Web Site's Menu Screens

Both sets of menu screens are designed to be as similar as possible. The visual appearance and the front end behavior of the two pages should be identical. The real difference between the two pages are in the messages displayed and the back end behavior of clicking on various links. Visually the menu screens are divided into four frames. The top left frame holds the logo graphics for the REPI web site. The top right frame displays the current title for the screen. As the user navigates through the different pages of the REPI web site, the title frame should reflect the current phase of the SEI Requirements Elicitation process model. The bottom left frame lists the current menu items for the screen. This frame itself is divided into three sections using the HTML *<TABLE>* tag. The top part of the table is used for the different phases of the process model. Generally this top part provides a link into each task of the current phase for the current user group. The middle part of this table is used for generic project management oriented tasks; these "tasks" are not part of the SEI Requirements Elicitation process model. Rather, as described further in Section 3.2.9, they are generic useful items which should make using the REPI web site easier. The last part of this table is used for generic

web site related links; specifically links such as "Main Menu," "Help," and "Logout" are provided. If the current page is one of the main menu pages then the "Main Menu" link is not displayed. If the current page is one of the help pages then "Help" link is not displayed. The "Logout" link, used to exit the REPI web site and redisplay the "Login Screen," is always displayed. The bottom right frame is used as the main display area for the actual forms needed in each of the tasks. The contents of this frame are completely dependent on the current phase, the current task and the type of user logged in to the REPI web site. Sections 3.2.3 to 3.2.7 describes these pages in detail, where each section is used for a different phase of the SEI's Requirements Elicitation process model.

The rest of this chapter doesn't describe the logo frame, title frame and the left frame. The logo frame is constant throughout the REPI web site. The title frame displays the current phase of the SEI's Requirements Elicitation process model. The left frame's contents have been just described above. The description of the REPI web site presented below, in Sections 3.2.3 to 3.2.9.4, refers only to the right frame of the given web page.

3.2.3 Fact Finding Phase

The Fact Finding phase of the SEI Requirements Elicitation process model examines the context of the project and the product to be developed. The client side members of the project examine their organization and their reasons for the project. The development side members of the project examine the technology and the domain of the product to be developed. Table 4 lists the mapping between the SEI tasks for the Fact Finding phase,

as described in [CHRISTEL 92], and the REPI web site tasks for the Fact Finding phase. The user oriented or client side tasks are listed first and then the development side tasks.

Table 4: SEI Compared with REPI for the Fact Finding Phase

Side	SEI's Tasks	REPI's Tasks
Client Side Tasks	Identify relevant parties.	Identify relevant people
	Determine operational and problem context	Describe the Problem
		Define Goals
		List Mission Scenarios
	Identify similar systems.	Identify similar systems.
	Perform context analysis.	
Development Side Tasks	Identify domain experts.	Identify domain experts.
	Identify domain and architectural models.	Identify domain models.
	Conduct technological surveys.	Conduct technological surveys.
	Assess cost/implementation constraints.	Assess constraints.

3.2.3.1 User's Tasks

3.2.3.1.1 Identify Relevant People: This task identifies the potential stakeholders of the project. Any Requirements Elicitation effort needs a set of people to work with. This web page can be used by project managers to identify people who could contribute to the success of the project. The people identified might be end users of the actual product to be built. They could be the customers or owners of the product to be built; people who actually authorized the project or people who are going to pay for it. They could be management level people, such as supervisors of end users, whose input could be of use

for the project's success. The web page displays a simple form for users to enter the stakeholder's name and basic contact information. It also has an option button group for categorizing the stakeholder into one of three pre-defined categories: "End User," "Customer," or "Management."

Other information about the stakeholder, such as their job description or title, could be of use in this page; but full information about each project member can be viewed from a separate page of the REPI web site. Specifically the "Project Member" menu item from the "Todo Tasks" page, described in Section 3.2.9.3.3, can be used to view all available information about each member of the project team.

3.2.3.1.2 Describe the Problem: This task allows the users to describe the perceived problem from their point of view. This allows each stakeholder to define their view point, possibly separate and contradictory to other view points. The web page presents a simple text area for the users to describe the problem using their own words.

3.2.3.1.3 Define Goals: In this tasks the users are expected to define the major goals to be achieved by the project. Each stakeholder can enter multiple goals by repeatedly using the form and giving a separate goal name each time. This provides the developers with a list of major areas of work to be done for the project. The web page presents a text box for the goal name and the text area for the users to enter the goal description.

3.2.3.1.4 List Mission Scenarios: Major scenarios for the product's use are identified in this task. The scenario has a name and general description associated with it. An event, action, reaction framework is used to list the steps of the scenario. The right side of the web page for this task is divided into three frames. The top frame has a text box and text area for the scenario name and its description. The middle frame holds the text boxes for the event, action, reaction framework. The bottom frame has three JavaScript buttons. The first button, labeled "One More Event Line," is to be used for creating another event line in the middle frame. If the scenario to be described has a complex or a long set of events, actions or reactions then more event lines can be created using this button. The second button, labeled "Enter Scenario" is to be used for signaling the finished scenario. The last button is the standard HTML form's clear button, used for clearing the contents of the form data fields.

3.2.3.1.5 Identify Similar Systems: This task is used to identify systems that are in some way, shape or form similar to the product to be developed. This allows users to identify potential sources of reuse materials, either in the form of analysis, design or perhaps the implementation level details itself. To be of any use, these potential sources have to be identified and categorized according to their usefulness. This task is used to identify these sources and list the similarities and differences between these systems and the product to be developed. As seen in Figure 32, the REPI web site's view for this task is presented in four frames on the right side of the web page. The top frame has a text box to identify the system for potential reuse. It also has a text area for a general

description of this system. The middle part is divided into two sub-frames. The left sub-frame lists the similarities identified between this system and the product to be built. The right sub-frame lists the differences identified between this system and the product to be built. The bottom frame has a text area to enter new information about this system. The left button, labeled "Enter Similarity" should be used if the newly entered information is a description of a similarity between the two systems. The right button, labeled "Enter Difference" should be used if the newly entered information is a description of a difference between the two systems. The middle button is the standard clear button used to clear the contents of the form data fields.

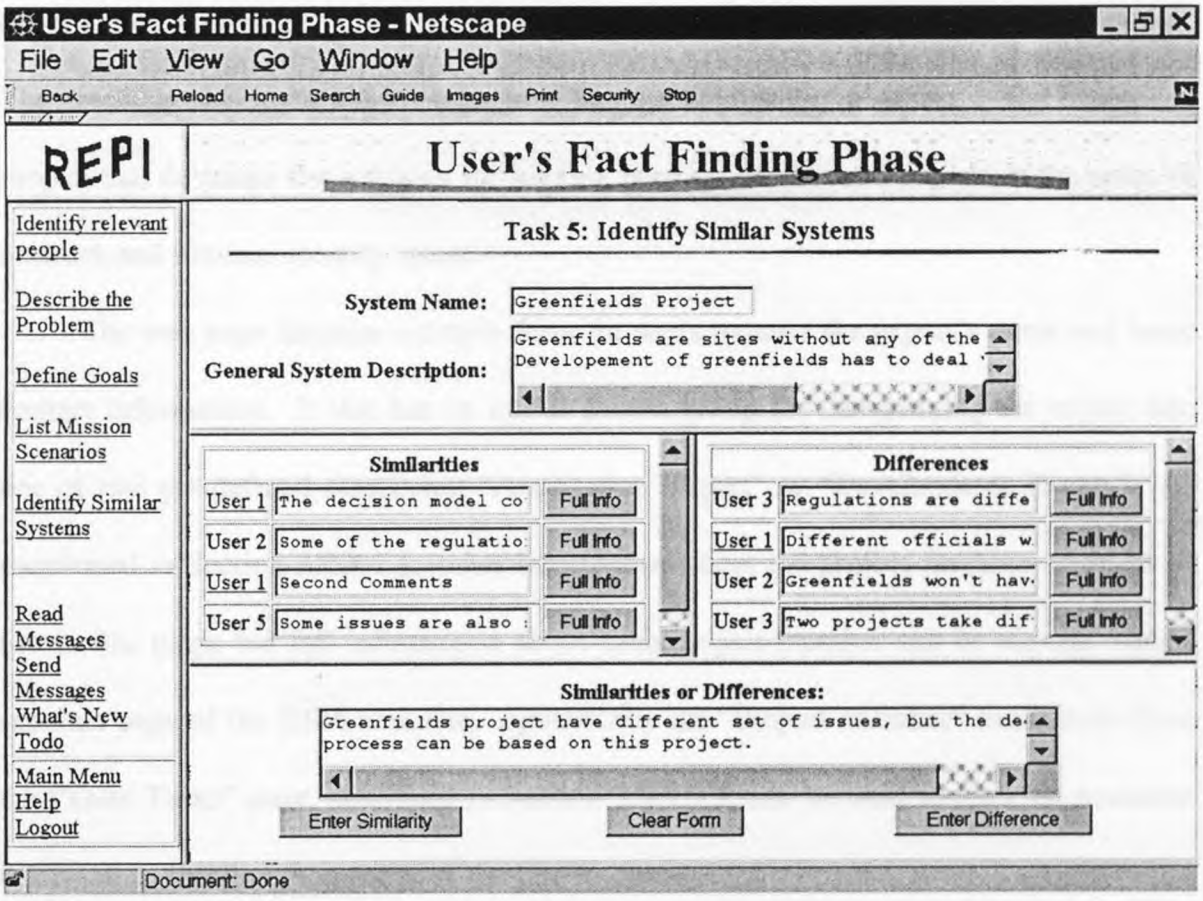


Figure 32: Task 5, "Identify Similar Systems," of the Fact Finding Phase.

3.2.3.2 Developer's Tasks

3.2.3.2.1 Identify Domain Experts: This task is similar to the user oriented task "Identify relevant people" executed in the same phase of the SEI's Requirements Elicitation process model. This tasks identifies the domain experts and the development experts for the project. The user oriented task identified people to contribute their specific needs for the product to be developed. The developer oriented task identifies people to contribute the specific needs for developing the product in a given domain. The application expert or the domain expert has knowledge in the general area of the product. For example a project that develops an application for a bank might require domain experts with knowledge about the laws that apply in the banking and finance industries. The development expert has knowledge in the product development areas. For example a project that develops the software for Internet banking might need experts in the areas of network and Internet security issues.

The web page displays a simple form for users to enter the expert's name and basic contact information. It also has an option button group for categorizing the expert into one of two pre-defined categories: "Application Expert" or "Development Expert." As mentioned in Section 3.2.3.1.1., other information about the project member could be of use in this page; but full information about each project member can be viewed from a separate page of the REPI web site. Specifically the "Project Member" menu item from the "Todo Tasks" page, described in Section 3.2.9.3.3, can be used to view all available information about each member of the project team.

3.2.3.2.2 Identify Domain Models: This task allows developers to identify the different models that will be used during the product's development. The REPI web site demo allows the developers to enter information about the domain model and the architectural model. The domain model refers to information about the product's general area. For example if the product to be built is an Automatic Teller Machine (ATM) software system for a bank then the domain model might contain information about how the ATM transactions are generally applied in the banking industry. The architectural model refers to information about the product design. Using the ATM software example again, the architectural model might contain information about using a client-server model for developing the front end software and the back end software. The REPI web site page for this task also contains a search button which is provided for the possibility that is information needs to be searched for. As the development process is just beginning such information might not be readily available for input. In case the information is available a file import button is also provided. This button allows the developers to provide a link into an external file that contains the necessary information.

3.2.3.2.3 Conduct Technological Survey: This task is used to enter technological survey information about the technologies that will be used during a product's development. For example if the project's purpose is to provide the software for Internet banking then this task can be used to enter information about the Internet security technologies such as Secure Sockets Layer (SSL), which is a security protocol needed for secure web applications.

The REPI web site demo provides a text box to enter a name for the technology survey. A survey should be named because multiple surveys, in different technological areas, might be needed for the product's development. The demo web page for this task also provides a text area to enter the survey information. A search button is provided to search for survey information and a file import button is provided to link into an external file that contains the survey information.

3.2.3.2.4 Assess Constraints: The purpose of this task is to gather information about the constraints imposed by the client side people. A constraint is an implied requirement that limits the design solution or implementation level choices of the system. The REPI web site displays a drop down box to select the constraint and provides a text area to enter the information about this constraint. For example the users might impose the choice of Microsoft Windows 95 as the software's platform. In such a case the developers need to gather information about this constraint's implications. One such implication might be the need for a developer who is familiar with the Windows 95 Application Programming Interface (API).

3.2.4 Gathering and Classification Phase

The Gathering and Classification phase of the SEI Requirements Elicitation process model is responsible for capturing and organizing a set of requirements for the product to be developed. The client side members of the project provide the requirements based on their needs and the development side members of the project classify the requirements

based on various attributes. The client side people are also allowed to provide information for the requirement's attributes. The development side people are also allowed to add requirements to the database. Table 5 lists the mapping between the SEI tasks for the Gathering and Classification phase, as described in [CHRISTEL 92], and the REPI web site tasks for the Gathering and Classification phase. The user oriented or client side tasks are listed first and then the development side tasks are listed.

Table 5: SEI Compared with REPI for the Gathering and Classification Phase

Side	SEI's Tasks	REPI's Tasks
Client Side Tasks	Get wish list.	List Requirements
		Add Requirements
Development Side Tasks	Classify wish lists.	Classify Requirements
		List Requirements
		Add Requirements

3.2.4.1 User's Tasks

3.2.4.1.1 List Requirements: This is not a task of the SEI Requirements Elicitation process model per se, rather it is a utility function useful in this phase of the process model and throughout the Requirements Elicitation effort. This "task" allows the users to list all the available requirements. It also allows the users to filter, sort, and define their own viewpoint into the requirements database. Several commonly useful pre-defined views are also provided here. Figure 33 shows the REPI presentation for this "task."

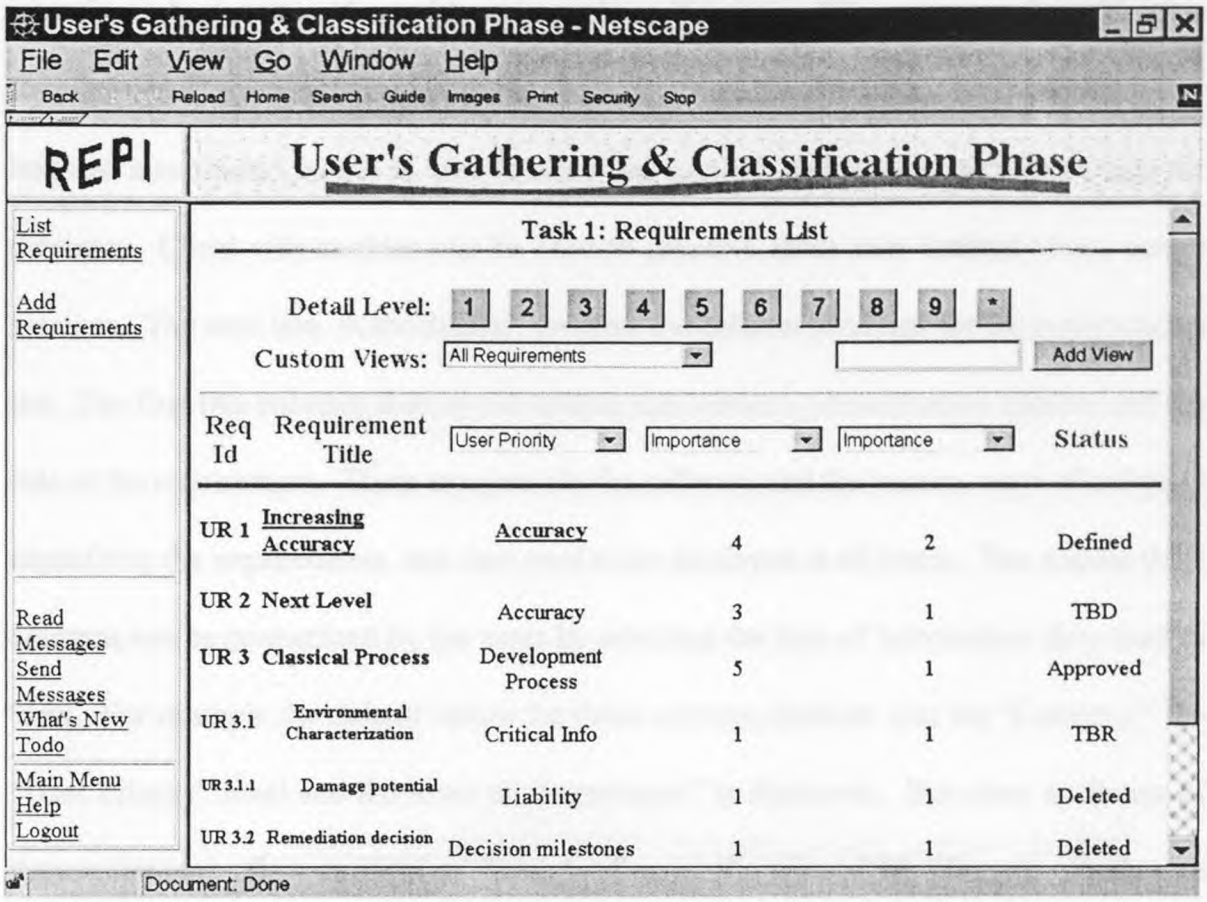


Figure 33: Task 1, "Requirements List," of the Gathering and Classification Phase

As shown in Figure 33, the right side of the web page is divided into two major parts. The top part defines the controls for the different views and the bottom part lists the actual requirements. Specifically the top row has 10 buttons which indicate the detail level for the displayed set of requirements. Level "1" indicates that only top level requirements such as "UR1," "UR2" or "UR3" are displayed. Level "2" indicates the next level requirements such as "UR3.1" or "UR3.2" are also displayed. Up to nine levels are defined using the first nine buttons. The last button, labeled with a "*" symbol, indicates that all the levels are to be displayed. On the second line of the display, a drop down list box is used to list the pre-defined views into the requirements database. The

users can select commonly useful views such as, "Undefined Requirements" or "Verified Requirements" to view these set of requirements from the database. Next to this, a text box and a command button is used to allow the users to define their own views into the database. Client side cookies can be used to preserve these user defined views across sessions. The next line, in the display, presents the column headings for the requirements list. The first two columns display the unique requirements identification number and the title of the requirement. These two provide the software and the human, ways of uniquely identifying the requirements, and thus need to be displayed at all times. The middle three columns can be customized by the users by selecting the type of information they need to view. For example the default values for these columns indicate that the "Category," the "User Priority" level and the level of "Importance" is displayed. But other attributes of the requirements such as level of understanding or the type of requirement can also be displayed in any of these three columns. The last column is used to display the current status for the given requirement.

3.2.4.1.2 Add Requirements: This is one of the most important tasks in the Requirements Elicitation process model. This is the primary means of adding requirements by the users. Client side users add or modify the requirements in this task. Besides the main paragraph describing the requirement, it should have several attributes associated with it. These attributes provide supplementary information about the requirement, its relationship to other requirements and assist in requirements management [KAR 96]. A unique method of identification is needed, both for the software and for

humans, to distinguish between the different requirements. Different types of categories are needed to classify requirements. The "Add Requirements" page allows user to provide both the main paragraph and these supplementary properties.

The requirements identification number, labeled as "Req Id," is used as the unique identification method by the software. The requirement title, a user provided text string, is used as the identification method by the humans. Each requirement is also categorized into several categories. Several pre-defined categories, based on the problem domain, could be defined. Users are also allowed to define their own categories. But if each and every user defines his or her own category it defeats the purpose of having categories. So some method of social control or software assisted security control needs to be provided, to allow only selected users to define new categories. Other users should be restricted to selecting a pre-defined category as they add new requirements. Each requirement can be held up to different levels of compliance. The compliance level defined here include "Mandatory," "Goal," "Objective," and "Optional." The exact semantics of these compliance levels has to be based on some external common understanding, such as a contract document. A requirement evolves through different stages. This evolution is reflected in the "Current Status" attribute with several status levels defined, such as "To Be Determined" (TBD), "To Be Reviewed" (TBR), "Defined," "Verified" and "Deleted." A requirement can be classified into several broad categories such as "Functional requirement," "Non-Functional requirement," or "Interface requirement." Some of the broad categories have several sub-categories such as "Performance requirement" or "User Interface requirement." The "Requirement Type" attribute is used for this classification

including the classification for information that is deemed to be a "Design Constraint" rather than an actual requirement. A given requirement has to be verified before it comes useful. Depending on the type of requirement and the available information on it different methods of verification can be used. The "Verified By" attribute is used to indicate the type of verification method desired for a given requirement. These methods are "Inspection," "Analysis," "Demonstration," and "Test." These attributes provide additional information about the requirement. Besides all these attributes each requirement has to have a description that defines the requirement itself.

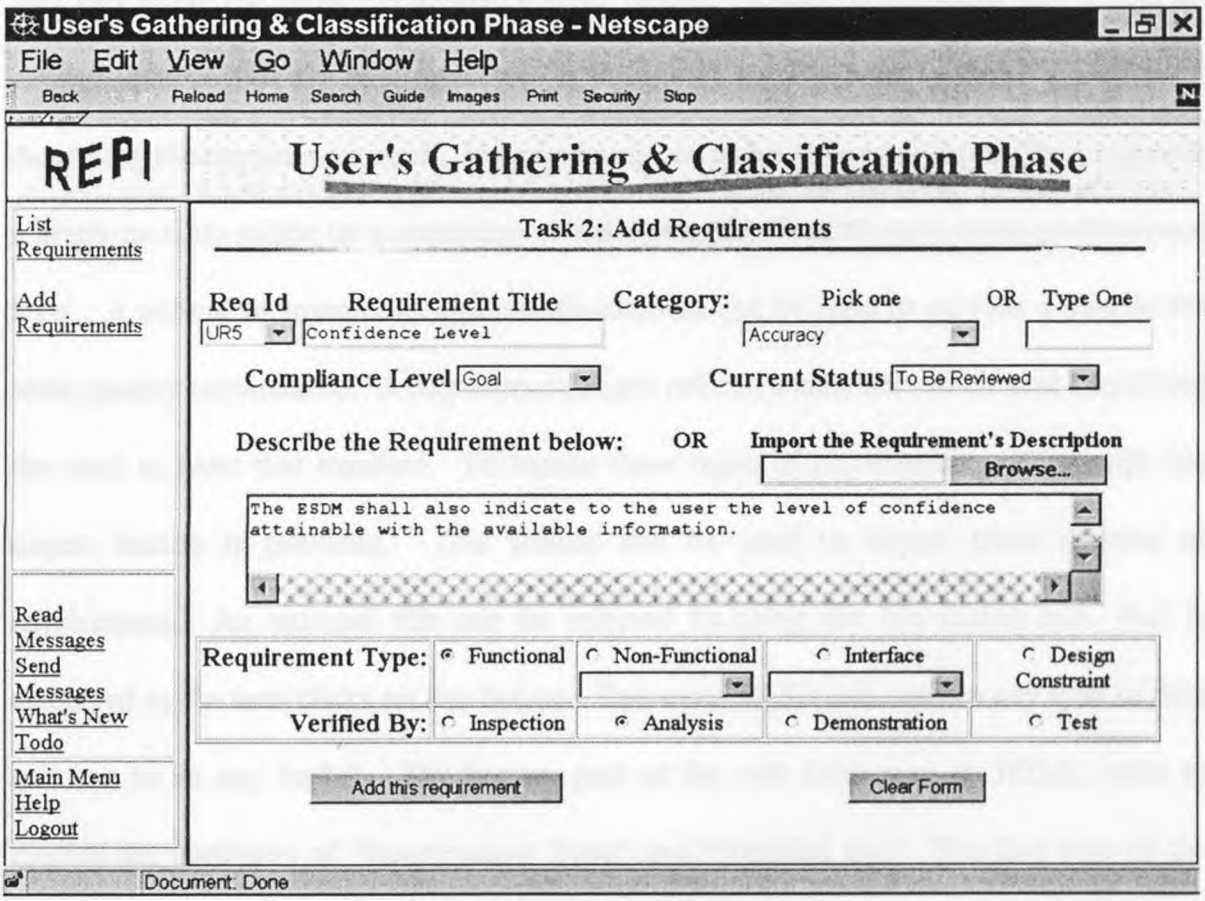


Figure 34: Task 2, "Add Requirement," of the Gathering and Classification Phase.

Figure 34 shows the REPI implementation for the "Add Requirement" task of the Gathering and Classification phase. This web page is designed to display all the properties that need to be entered for a requirement to be fully defined. The web page first uses a drop down list box to list the next available requirement identification number and provides a text box for the user to enter the requirement title. To the right side of this, another drop down list box is used to list the pre-defined categories. A text box is provided for the users to enter their own category. The next row provides the list of compliance levels and current status indicators. Either an end user or a management level user can select the proper compliance level needed for this requirement and define its current status. Next a text area box is presented for the user to enter a paragraph describing the requirement itself. Requirements can come in several forms, for a example a graph or table might be a requirement specifying the need to meet some performance level. A picture or some other multimedia element can be used to provide a sample for some quality requirement. A requirement might refer to a standards document specifying the need to meet that standard. To handle these types of requirement a JavaScript file import button is provided. This feature can be used to import these sources of requirement. An external file can be referred to using the file dialog box, that is displayed as the user clicks on this button. This external file can contain any type of data and can be in any format. The bottom part of the web form uses an HTML table to display the attributes of "Requirement Type" and "Verified By." The first row of the table displays the different types of requirements, including the sub-types for the "Non-Functional Requirement" type and the "Interface Requirement" type. A grouped radio

button is used to select the type of requirement and the drop down list box is used to select the proper sub-types. The second row of the table displays the different types of verification methods available. Once again, a grouped radio button is used to select the verification method. The two buttons on the bottom of the form are used to either enter the requirement into the database or to clear the HTML form.

The details of the above attributes differ from one source to another. For example, in the area of requirement categorization, several different possibilities are listed: "Program Requirement" versus "Product Requirement" and "Primary Requirement" versus "Derived Requirement" [KAR 96] and [HARWELL 93]. A Requirement application attribute identifies the object of a requirement with several different types of parameters [HARWELL 93]. The "Product Parameter" of a requirement can be subdivided into "Qualitative" and "Quantitative" parameters. The "Program Parameter" of a requirement can be subdivided into "Task," "Compliance Evaluation," and "Regulatory" parameters. For a requirement's compliance level, [HARWELL 93] uses the three values "Mandatory," "Guidance," and "Information," while [KAR 96] uses the two values "Mandatory" and "Goal or Objective." A given requirement can have several other attributes that are not included in this demo version of the REPI web site. Some of these attributes, described in [KAR 96], are "Allocated to," "Source," "Verification documents," and "Change notices." The "Allocated to" attribute allocates each requirement to a lower level component of the product to be built. "Verification documents" are documents which describe verification plans and procedures, such as a test plan document. "Change notices" refer to information recording the history of the

requirement as it evolves. These attributes could be added to the prototype version and the support for existing attributes can be improved upon.

3.2.4.2 Developer's Tasks

3.2.4.2.1 Classify Requirements: This task allows developers to properly categorize the requirements entered by the users. The purpose of this task is to produce a detailed requirements hierarchy. While the requirements can be categorized as they are added, a separate task is created for this purpose because full information about the requirements are not available at once, and also because most end users are not properly qualified to classify requirements based on their type such as, "Functional," "Non-Functional," or "Interface." Most end users are also not qualified to judge a requirement as a "Design Constraint" or as an actual requirement. Developers might need to consult with users if they need to change a requirement's category as they are producing a requirements hierarchy. A requirements might also go through several versions and slowly evolve from an undefined stage, with just the title given, to a fully defined and categorized stage; such an evolution might prevent the requirement from being properly classified at the beginning. Another usefulness of this task is that it presents information in a manner which makes it easy to properly categorize the requirements. All available information about each requirement can easily be viewed by selecting the requirements from the drop down box. So developers are assisted in categorizing requirements because they can view already categorized requirements as an example to categorize new requirements. This

helpful feature is not provided in the "Add Requirement" task, described above, because that task's web page had a different design goal.

The REPI web site demo presents this task's web page in three different parts separated by a horizontal line across the right side of the screen. The top part displays a drop down box containing a list of all available requirements. Next to it a category name is displayed if one has been defined already. The middle part of the display has a list of user's description and comments on the selected requirement. This page's display is limited to one line for each user; clicking on the button labeled "Full Info" should display the full comment as entered by that user. The bottom part of the screen displays the different attributes of a requirement as discussed above in Section 3.2.4.1.2.

3.2.4.2.2 List Requirements: The Developer's task "List Requirements" is exactly the same as the User's task "List Requirements." Since the "List Requirements" task and the REPI implementation of this task is described above, in Section 3.2.4.1.1, it will not be described here again.

3.2.4.2.3 Add Requirements: The "Add Requirements" for the Developer's side of the REPI web site is the same task described in the User's side of the REPI web site. So please see Section 3.2.4.1.2 for this task's description and information about the REPI implementation.

3.2.5 Evaluation and Rationalization Phase

The Evaluation and Rationalization phase of the SEI Requirements Elicitation process model is responsible for exposing inconsistencies in the gathered information and it is also responsible for "determining why the information has been expressed as a requirement" [MILLER 93]. Table 6 lists the mapping between the SEI tasks for the Evaluation and Rationalization phase, as described in [CHRISTEL 92], and the REPI web site tasks for the Evaluation and Rationalization phase. The user oriented or client side tasks are listed first and then the development side tasks are listed.

Table 6: SEI Compared with REPI for the Evaluation and Rationalization Phase

Side	SEI's Tasks	REPI's Tasks
Client Side Tasks	Perform abstraction to answer questions of the form "Why do you need X?"; this in effect moves from statements of "how" to statements of "what."	Perform Abstraction
	Capture rationale to support future requirements evolution.	Capture Rationale
Development Side Tasks	Perform risk assessment.	Risk Assessment
		Feasibility Analysis
		Cost/Benefit Analysis

3.2.5.1 User's Tasks

3.2.5.1.1 Perform Abstraction: This tasks allows users to describe their requirements in more detail. The main purpose of this task is to answer the question: "Why do you

need this requirement?" To fulfill this purpose the web page allows the users to display each requirement, along with its description, and answer this question. The form for this task is displayed in three frames on the right side of the web page. The left frame lists the short titles for the requirements, using a list box. The top frame displays the full title, category and the description for the selected requirement. The bottom frame provides a text area for the user to answer the question about the displayed requirement.

3.2.5.1.2 Capture Rationale: This tasks is similar to the previous task, in the sense that this task also requires the user to enter more information about each requirement. In this task, the user is asked to rationalize about the given requirement. The rationale for a requirement provides data which support the requirement. "The supporting data may include the reason or reasons a requirement is needed; any assumptions made at the time the requirement was formulated . . ." [KAR 96]. The web page for this task is very similar to that of the previous task. But this task's web page has an import button and its associated text box displayed along with the text area. A requirements rationalization can come in any form including a graph or other multimedia item describing the rationale for a given requirement. The JavaScript file import button, labeled "Browse" and its associated text box can be used to import external items as the source for a given requirement's rationale. Clicking on the "Browse" button brings up a standard file dialog box for the given graphical user interface. This dialog box can be used to locate the source document to describe the given requirement's rationale. Figure 35 shows this file

dialog box as seen on the Netscape Navigator browser Ver. 4.03, running on the Microsoft Windows 95 platform.

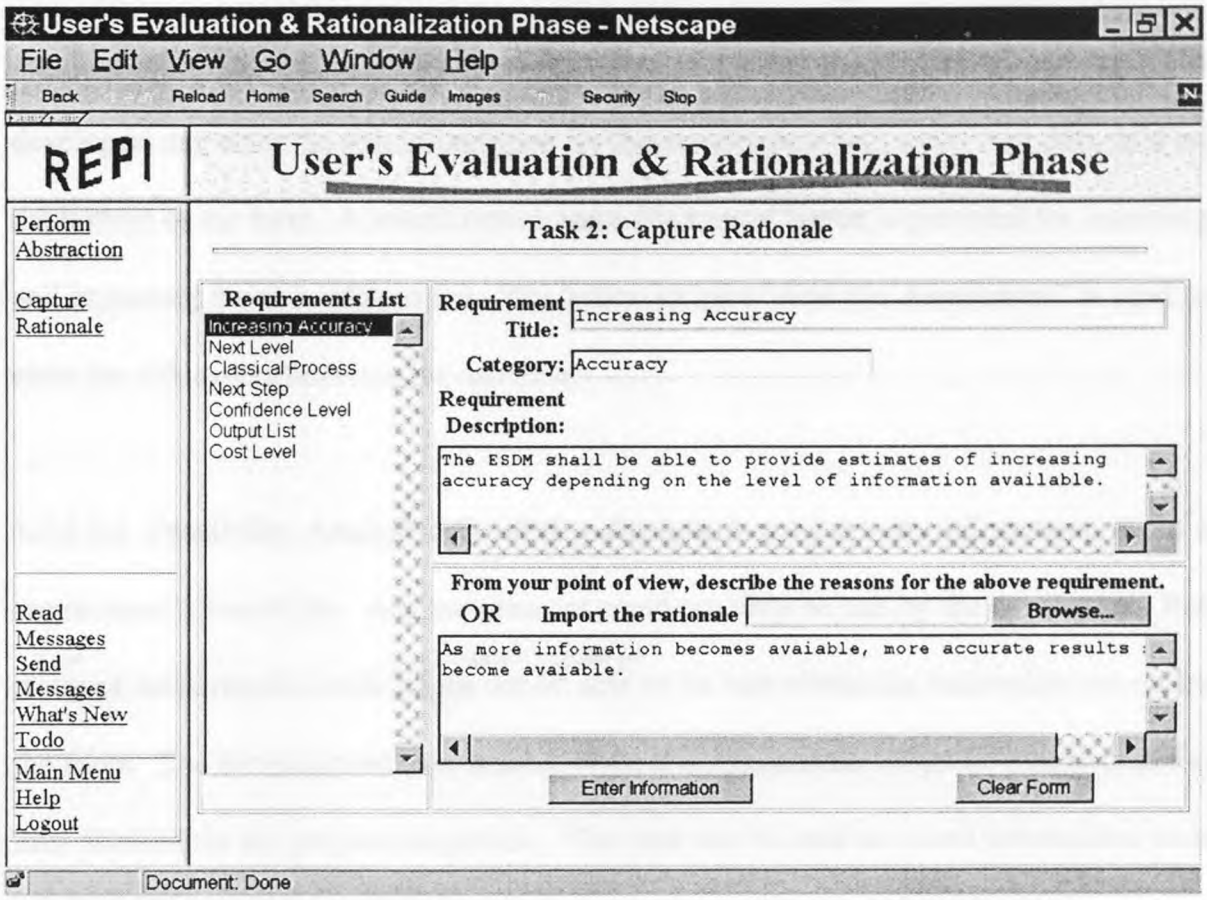


Figure 35: Task 2, "Capture Rationale," of the Evaluation and Rationalization Phase

3.2.5.2 Developer's Tasks

3.2.5.2.1 Risk Assessment: This task allows developers to keep track of the risks associated with each requirement. Using the example of Internet banking, the requirement might state the need for the web application to be compatible with older browsers. But if these older browsers do not support SSL or any other form of secure

transactions a certain amount of risk is associated with the requirement of using older browsers. Risks associated with such requirements can be described in this task.

The REPI web site displays the requirements list using a list box. The selected requirement's description is displayed right next to it using the *<TEXTAREA>* tag. The developers can enter the risk information for this requirement into a text area data field on the bottom of the form. A search button and a file import button is provided for searching and importing the risk information. The button labeled "Add this Assessment" is used to enter the risk assessment into the database.

3.2.5.2.2 Feasibility Analysis: Feasibility Analysis is used to enter information about a requirement's feasibility. Any requirement could possibly be met by the developers. But many of these requirements might not be able to be met within the constraints set up by the users. The development time needed to meet a requirement might be longer than the time allowed for the project completion. This task can be used to record information such as these, stating that a certain requirement can not be met. Or this task could be used to record information that states the conditions under which a given requirement can be met.

The REPI implementation for this task is very similar to that of the previous task, described above in Section 3.2.5.2.1.

3.2.5.2.3 Cost/Benefit Analysis: Costs associated with a requirement and the benefits derived from the requirement can be described in this task. The REPI implementation for

this task is very similar to that of "Risk Assessment" task described above, in Section 3.2.5.2.1.

3.2.6 Prioritization and Planning Phase

The Prioritization and Planning phase of the SEI Requirements Elicitation process model determines "the relative importance of each requirement and the relative order the requirements should be addressed in" [MILLER 93]. Table 7 lists the mapping between the SEI tasks for the Prioritization and Planning phase, as described in [CHRISTEL 92], and the REPI web site tasks for the Prioritization and Planning phase. The user oriented or client side tasks are listed first and then the development side tasks are listed.

Table 7: SEI Compared with REPI for the Prioritization and Planning Phase

Side	SEI's Tasks	REPI's Tasks
Client Side Tasks	Determine criticality.	Prioritize the Requirements list
Development Side Tasks	Prioritize requirements based on cost and dependency.	Prioritize Requirements
		Plan incremental development stages
		Identify architectural models

3.2.6.1 User's Tasks

3.2.6.1.1 Prioritize Requirements: This task is used to prioritize the requirements by considering which requirements are important or which requirements are more difficult and thus have to be dealt with more thoroughly. The users are required to set a priority level for each of the requirement listed, indicating its importance for the project from their point of view. The users are also required to judge the level of understanding obtained on the given requirement by the development team and by the users themselves. This indicator can be used to see which requirements are less understood and thus require more study or more detailed explanation. The REPI web site lists the available requirements along with two drop down boxes to set these indicators. The first set of drop down boxes are in the column marked "Priority Level," indicating the level of importance given to the requirement by the users. The second set of drop down boxes are in the column marked "Level of Understanding," and this indicates the understanding obtained by the project team on the given requirement.

3.2.6.2 Developer's Tasks

3.2.6.2.1 Prioritize Requirements: This task allows developers to prioritize the set of requirements and is very similar to the "Prioritize Requirements" task described for the User oriented task. The developers prioritize the requirement along two attributes: cost level and dependence level. The cost level describes the costs associated with a requirement; the assumption being that a more complex requirement costs more in development time than the time required for a simpler requirement. The dependence

level describes the number of related requirement associated with a given requirement. If a change in any requirements affects other requirements than these other requirements refers to the requirement. If a change in other requirements affects this requirement then this requirement is referred to by these other requirements. As the dependence level increases the cost of meeting and the cost of not meeting that requirement increases.

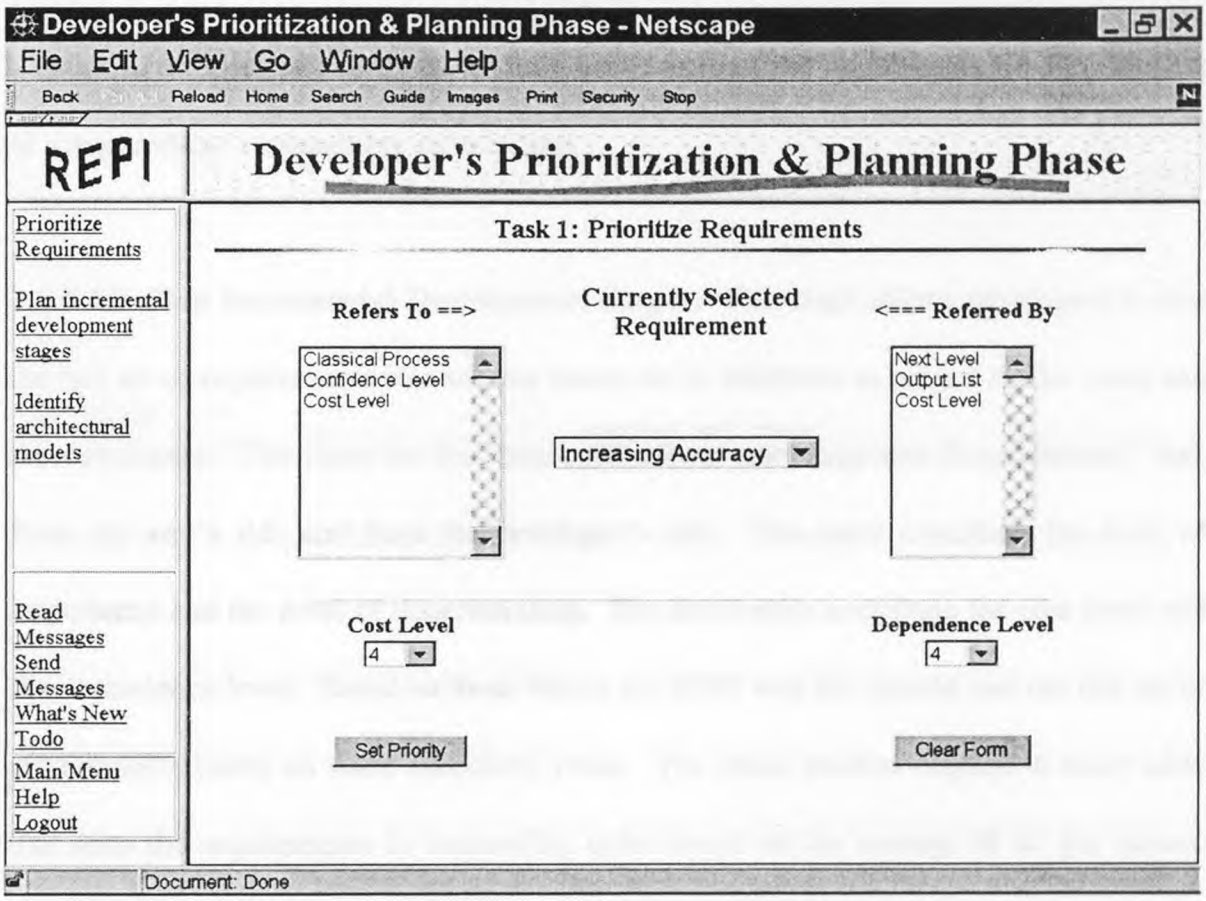


Figure 36: Task 1, "Prioritize Requirements," of the Prioritization and Planning Phase

Figure 36 shows the implementation from the REPI web site demo. The REPI web site's view for this task makes it easy to look at a requirement's dependents. A drop down list box is displayed in the center, listing the full set of requirements in the

database. To the left of this full requirements list, another list box displays the subset of requirements that refers to the selected requirement from the center list box. To the right of the full requirements list, another list box displays the subset of requirements that is referred to by the selected requirement from the center list box. The semantics for "Refers To" and "Referred By" are described above. So in effect these two list boxes provide the full influence the selected requirement has on the product. Based on this display, developers can easily set both the cost level and the dependence level by looking at a requirement's traceability information.

3.2.6.2.2 Plan Incremental Development Stages: This tasks allows developers to sort the full set of requirements into subsets based on its attributes as judged by the users and the developers. The input for this task comes from the "Prioritize Requirements" task from the user's side and from the developer's side. The users contribute the level of importance and the level of understanding. The developers contribute the cost level and the dependency level. Based on these values the REPI web site should sort the full set of requirements based on some combined value. The demo version displays a static table that sorts the requirements in descending order based on the average of all the values. Additional support for sorting and filtering the requirements should be provided in the prototype version. Figure 37 shows this sorted view into the requirements database.

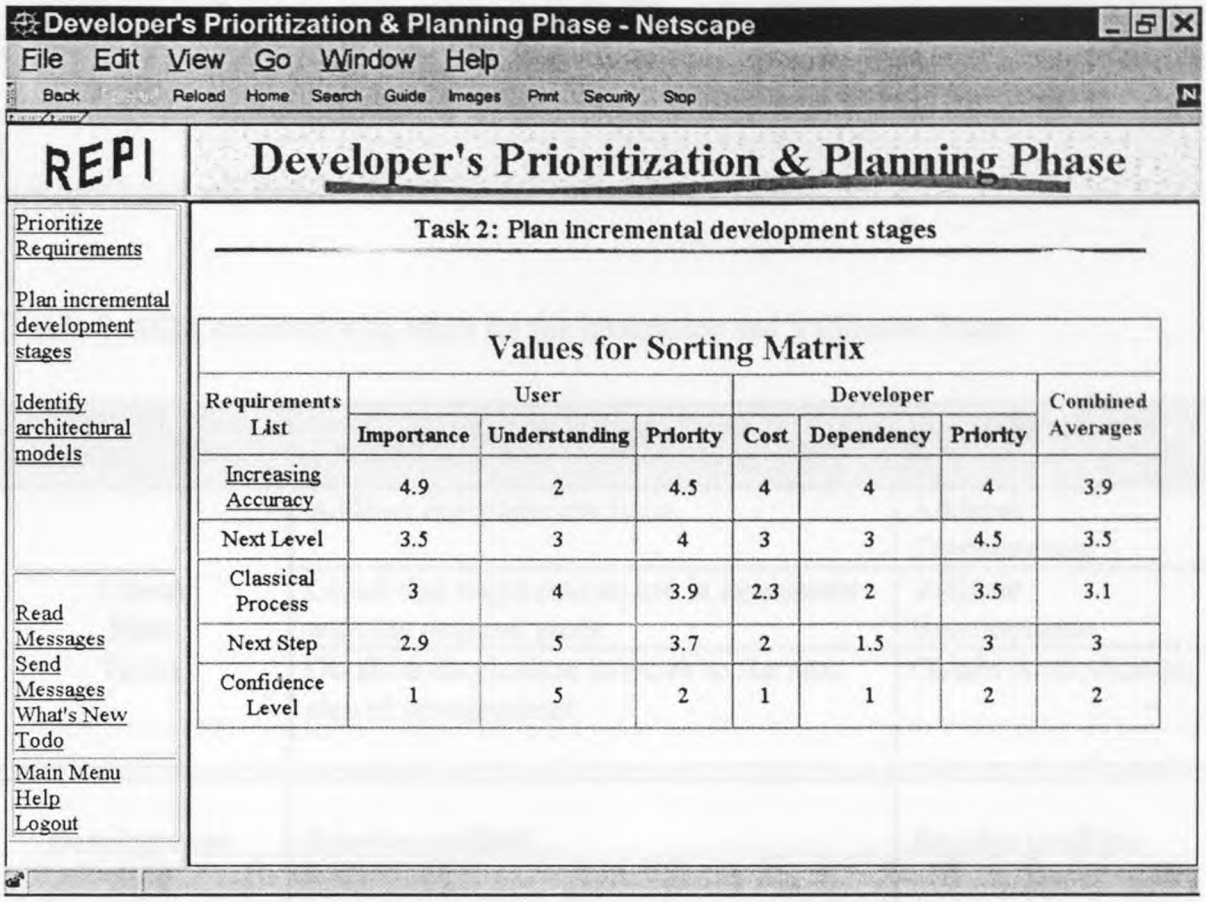


Figure 37: Task 2, "Plan development stages," of the Prioritization and Planning phase

3.2.6.2.3 Identify Architectural Models: This task requires developers to identify architectural models that support the incremental development stages identified in the previous task. The REPI web site demo's implementation is very similar to Section 3.2.3.2.2's description for the "Identify domain models" page.

3.2.7 Integration and Validation Phase

The Integration and Validation phase of the SEI Requirements Elicitation process model is responsible for determining the validity of the gathered information and it is also responsible for obtaining missing information. Table 8 lists the mapping between the SEI

tasks for the Integration and Validation phase, as described in [CHRISTEL 92], and the REPI tasks for the Integration and Validation phase. The user oriented or client side tasks are listed first and then the development side tasks are listed.

Table 8: SEI Compared with REPI for the Integration and Validation Phase

Side	SEI's Tasks	REPI's Tasks
Client Side Tasks	Address completeness issue	Address Completeness
	Check that requirements are in agreement with the original goals	Validate Requirements
	Obtain authorization to move to the next step of development	Obtain Authorization
Development Side Tasks	Resolve conflicts	Resolve conflicts

3.2.7.1 User's Tasks

3.2.7.1.1 Address Completeness: This task is needed to address any requirements that might not have been completely defined in the earlier phases of the Requirements Elicitation process. For example unknown or less understood requirements can be created and marked, during the "Gathering and Classification Phase," as "To Be Determined." These "TBD" requirements, as it is commonly listed, have to be eventually defined before the requirements stage of the development process is finished. The "Address Completeness" task of the "Integration and Validation Phase" is used for this purpose. The REPI web page for this tasks divides the right side of the screen into two

frames. The top frame lists the requirements that have been marked as "To Be Determined" and clicking on these requirements will activate the bottom frame displaying the "Add Requirements" form as described above in Section 3.2.4.1.2.

3.2.7.1.2 Validate Requirements: Before a set of requirements becomes useable, they have to be verified and validated. This task is used for the purpose of validating requirements and verifying that they are in agreement with the originally stated goals for the project. The REPI web site's view into this task is a form divided into three parts using the HTML table tag. The top part uses a list box to list all the available requirements and next to it, more information about the selected requirement is displayed. The bottom part uses another list box to list all the available goals of the project and next to it, more information about the selected goal is displayed. The user is required to select a requirement from the top part and view all the listed goals on the bottom part. If the selected requirement is in agreement with the listed goals, this requirement is considered to be valid as far as the goals are concerned. The JavaScript button on the middle part of the form can be clicked on to indicate the selected requirements' validity.

3.2.7.1.3 Obtain Authorization: This task is a simple step used to indicate the finished status of the Requirements Elicitation process. By "signing" this form the client side users indicate that the requirements have been properly elicited from them and that they authorize the developers to proceed to the next step of the development process. The REPI web site displays a text box for the users to enter their name used as an indication

of their "signing" the form. A JavaScript file import button is also provided on this form. This can be used to import a digital signature that can provide more security than a simple text box. The two buttons on the bottom of the form are used for either authorizing the developers to proceed or to deny their approval. If the users deny their approval then one more iteration through the Requirements Elicitation process is needed.

3.2.7.2 Developer's Tasks

3.2.7.2.1 Resolve Conflicts: In this task, the developers validate requirements and resolve any conflicts found in them. The REPI web site displays all the comments made by different users about the selected requirement. Based on these comments the developers are required to assess the validity of the requirement and resolve any conflicts that might arise from the different viewpoints presented by the users.

The REPI web site demo uses a drop down list box to present a list of requirements tagged as unverified or unresolved. Next to this the requirement's category is displayed and its description is displayed below that. A table is used to display the list of users who commented on this requirement. The left column of the table displays the user id, the next column displays the date and time the comment was made. The last column displays the actual comment made by that user. Below this table, a text area is provided where the developer can resolve any conflicts found.

3.2.8 Information Pages of the REPI Web Site

This section of the chapter describes the web pages of the REPI web site that provides detailed information for a given object. The first sub-section describes the "Requirements Information page" which displays all available information about a given requirement. The second sub-section describes the "Category Information page" which displays all available information about a given category. The third sub-section describes the "User Information page" which displays all available information about a given user. These pages are not directly related to the different phases of the SEI's Requirements Elicitation process model; rather they provide detailed information about the objects that are relevant in these phases.

3.2.8.1 Requirements Information Page

This page displays all available information about a given requirement. Both the client side people and the developer side people are presented with the same information, regardless of the information's origins. The REPI web site demo page is divided into four sections after the requirement's title is displayed on top of the frame. The first part uses a table to display the basic attributes about the given requirement. The first column displays the requirement's category, the next column displays the type of requirement, and the next one displays the verification method to be used on this requirement. The last but one column displays the compliance level for this requirement and the last column displays the requirement's current status. Whenever any information is not yet available, a blank space is displayed in that data cell. Clicking on the requirement's category should

display more information about that category as described below, in Section 3.2.8.2. The second part of the page uses another table to display the various priority levels for the requirement: importance, understanding, user priority, cost, dependency, and the developer priority. The next part of the page displays the four types of additional information entered for each requirement. The top left cell of this table displays the rationalization information about the requirement; this information is entered by the users in the second task of the "Evaluation and Rationalization Phase" of the SEI Requirements Elicitation process model. The top right cell of this table displays the risk assessment information for the requirement; this information is entered by the developers in the first task of the "Evaluation and Rationalization Phase." The bottom left cell displays the feasibility analysis, from the second task of the "Evaluation and Rationalization Phase." The bottom right cell displays the cost and benefits analysis, from the third task of the "Evaluation and Rationalization Phase." This part of the page is shown in the first part of the Figure 38.

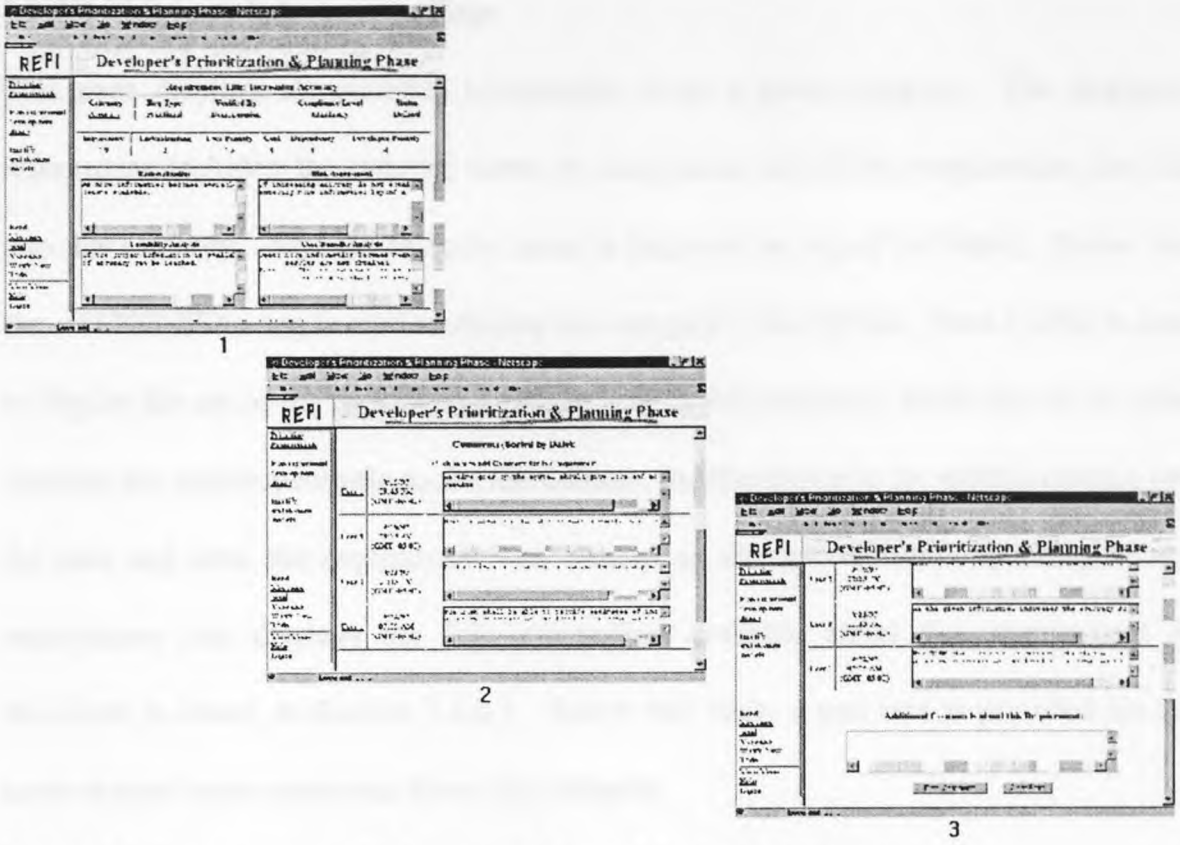


Figure 38: "Requirements Information" Page of the REPI Web Site.

The second part of Figure 38, shows the list of comments made by different users about this requirement. The left hand column displays the user id, the middle column displays the date and time the comment was made, and the last column displays the actual comments made by a given user. Clicking on the user id for a given project member should display all available information about that user, as described below in Section 3.2.8.3. The last part of Figure 38, shows a text area provided for users to enter comments about this requirement.

3.2.8.2 Category Information Page

This page displays all available information about a given category. The displayed information includes: the category name, its description, and all the requirements that fall into this category. First the category name is displayed on top of the frame. Below that the *<TEXTAREA>* tag is used to display the category's description. Next a table is used to display the set of the requirement that belong to this category. Each row of the table contains the requirement title on the left column, its description in the middle column and the date and time the requirement was created on the last column. Clicking on the requirement title displays the full information available about that requirement as described in above, in Section 3.2.8.1. Below this table, a text area is provided for the users to enter more comments about this category.

3.2.8.3 User Information Page

This page displays all the contributions made by a given project member. The page starts with the basic information about the person, such as the name and the email address. As described below and mentioned above, full information about a given project member can be accessed from the "Project Members" menu item from the "Todo Tasks" screen of the REPI web site. Following the basic information about the person, a list of requirements added by this person is displayed. For this requirements list, a table is used where the first column holds the requirement title, the second column holds the description and the last column holds the date and time of the requirement's creation. Clicking on the requirement title should display the full information available on that requirement, as

described above in Section 3.2.8.1. Below this table, all the comments added by this person, to the various requirements, are listed. This table also displays the requirement title on the first column; but the second displays the user's comments instead of the requirement's description. The last column displays the date and time the comment was entered into the database. Below this table, terms added by this person are displayed. In this table the first column refers to the term name, the second column refers to the term's definition and the last column refers to the date and time the term was created. Clicking on the term name should display the "List Terms" page of the "Todo Tasks," as described below in Section 3.2.9.3.1.

3.2.9 Other Components of the REPI Web Site

These items are not part of the SEI Requirements Elicitation process model or the framework; rather they are helpful and useful items that are needed for an easier and a more productive use of the REPI web site. These items are also not divided between client side user and development side users, unlike the tasks from the Requirements Elicitation process model's different phases. These are common items assessable and helpful for members of both the client side environment and the development side environment. Many web based applications and standard client server applications do provide these services and are implemented far better than it is shown in this demo web site. In a fully implemented prototype, the users of the web site will be better served by replacing these components with web based applications that are specifically designed for these functions. For the sake of completeness these components are included as part of

the REPI web site demo. These components will be briefly described below for the purpose of mentioning their usefulness to the Requirements Elicitation process.

3.2.9.1 Read and Send Messages

Studies such as [AL-RAWAS 96] have shown that unregulated, informal, and interpersonal communication channels are as necessary as regulated, formal and group communication channels. Information sharing and exchange in a group is based on these informal channels as well as formal channels such as written documents or formal meetings. For this purpose an email type messaging facilities is provided as part of the REPI web site. This part of the REPI web site is presented as two menu items and thus two sets of web pages. The "Read Messages" page is used to retrieve and read messages sent to the individual users or to their groups. The "Send Messages" page is used to send these messages to the individual members of the project team or to whole groups of people involved in the project.

The "Read messages" page lists the current messages for the given user on the top frame. Clicking on the message's subject line displays the actual message on the bottom frame. The bottom frame also provides three buttons for different types of actions that might be taken regarding this message. The "Send messages" page provides a form for the user to enter the message contents and select the various attributes associated with a given message.

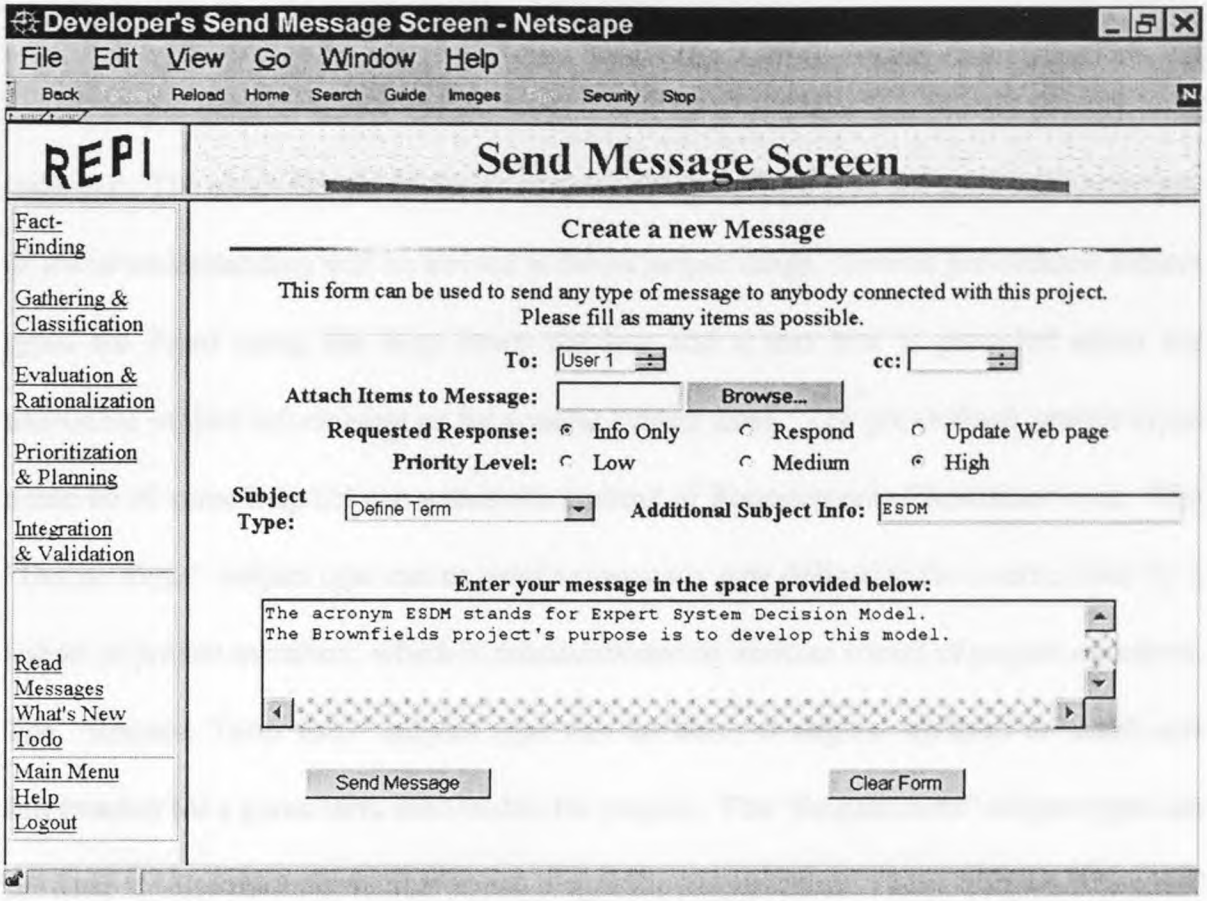


Figure 39: "Send Messages" Page of the REPI Web Site

As seen in Figure 39, a message has several attributes that might be of help in the context of the REPI web site usage as a group "meeting" place for the purpose of eliciting requirements. First the "To:" list box and the "cc:" list box are displayed to select the message's destination. Then a file import button and its associated text box is provided to attach items to the message. Then a set of attributes is provided that details the purpose and type of message to be sent. The first attribute, labeled "Requested Response," indicates the type of response requested by the sender of the message from the receiver of the message. The "Info Only" option indicates that no response is requested, the "Respond" option indicates that a response message is requested and the "Update

Web page" option indicates that the response should be in the form of updating the specified web page with new information. The second attribute is a generic priority level indicator. The exact semantics for its usage is not described; it is assumed that some type of social understanding will be arrived at for its proper usage. Several pre-defined subject types are listed using the drop down list box and a text box is provided either for additional subject information or for generic subject lines. The pre-defined subject types could be of some help for use within the context of Requirements Elicitation work. The "Define Term" subject type can be used to request a new definition for a term, used by a subset of project members, which is misunderstood by another subset of project members. The "Request Term Info" subject type can be used to request updated or additional information for a given term used within the project. The "Requirement" subject type can be used to discuss information about a specific requirement. The "Project Member" subject type can be used to discuss information provided by a specific project member. The "Project Status/Progress" subject type can be used to discuss the current status or current progress level of the project or any of its phases. The generic subject type "Project" can be used to discuss something related to the project itself, perhaps the process/work/data flow or the management aspects of the project. The generic subject type "Web Site" can be used to discuss information about the REPI web site itself, perhaps suggestions for improvements or request of help on its usage. The subject type of "(Other)" is used to indicate that additional subject information is to be used as the main subject heading in the message list.

3.2.9.2 What's New Screen

The "What's New" page displays a list of objects that have changed recently. In a long term project or in a project with many members it becomes difficult to keep track of all the changes made to the requirements database or all the changes happening during the project. This page can be used to automatically list the objects that have changed since the user's last login. This should provide a user with a ready reference point to see which items he or she has to look through and respond to. This should help users manage "information overload" that might happen in a given long term project. It also helps users keep track of items, if they are working on multiple projects.

The first item on this page displays the newly arrived or unread messages sent to a given user. Clicking on this number should provide the user with a list of these messages. Similarly the second item displays the newly arrived or unread messages sent to a given user's group. Groups can be anything the project managers decide it to be. In the REPI web site, it is represented just as a user name would be; instead of referring to an individual user, a group name refers to a set of users. Groups can be created along functional areas of the product to be built or across functional areas based on the type of user's expertise. For example a "User Interface" group can be created for people responsible for defining user interface requirements. This group of people might consist of user interface design experts from the development side and it might consist of end users whose input in this area is considered to be valuable. The next item on this page displays the set of newly created or modified requirements. If the number of requirements is large, than a constant notification of changed requirement will be of use

to users who are infrequent visitors to the web site. Similarly the next item on the page displays the newly created or modified terms and their definitions. The next item is displayed whenever updated schedule information is made available by project managers. The last item on the page displays a list of project related reports, such as progress reports or summary reports, as they are created by project members and made available on the web site.

3.2.9.3 Todo Tasks

3.2.9.3.1 Define and List Terms: Project members from different disciplines and backgrounds will bring in different types of knowledge with various levels of expertise. In any such multi-disciplinary project a subset of project members will use terms and definitions that might not be commonly understood by all members of the project. Terms used by one project member might be misunderstood by people with different backgrounds. Even people from the same background might understand it in a different manner. This set of "Todo tasks" are useful in such situations. The "Define Terms" and "List Terms" pages of the REPI web site, respectively, allows project members to define and view terms used in a project. Using these "tasks," project members across disciplines and backgrounds can come to a common understanding on the various terms to be used in the project.

The REPI web site's page for the "List Terms" task uses a table to list the term names on the left hand column and its definition on the right hand column. The "Define Terms" page provides a text box for the term name to be entered and a text area for the

term's definition to be entered. The "Enter Term" button is to be used to update the terms database with a newly created or a modified term definition.

3.2.9.3.2 Project Schedule and Status Reports: In any project, even in a distributed Requirements Elicitation effort, the need for project scheduling and planning exists. The REPI web site demo page "Project Schedule" displays a static image of a chart depicting the planned schedule for the different phases of the SEI's Requirements Elicitation process model. In an actual prototype web site, this page should display a dynamic image that is updated as the scheduling and planning information is made available. An imagemap can be imposed on these charts where clicking on an allocated time unit displays more detailed information for the activities of that time unit. Perhaps different charts can be displayed based on user id or group id. Web based applications that are specifically designed for scheduling and planning activities can also be linked for this part of the REPI web site.

The "Status Reports" page of the REPI web site provides a common location for all the reports and other documents produced in a project. The demo web page uses a table to display a set of links that refer to a Microsoft Word document file called "STATUS.DOC." The demo page provides a link for one status report and one documentation report for each of the five phases of the Requirements Elicitation process model. Within each phase the reports are sorted by date. The behavior produced by clicking on these links is entirely dependent on the browser's configuration settings and Internet Media Type associated with the file extension "DOC."

3.2.9.3.3 Project Members: Figure 40 displays the "Project Members" page of the REPI web site. In this page, available information about each project member is displayed. This provides the opportunity for all members of the project to get to know each other. A page such as this is necessary because in a distributed environment such this, all project members might not necessarily meet each other. Knowing basic information such as their job title and description should make working with these people easier.

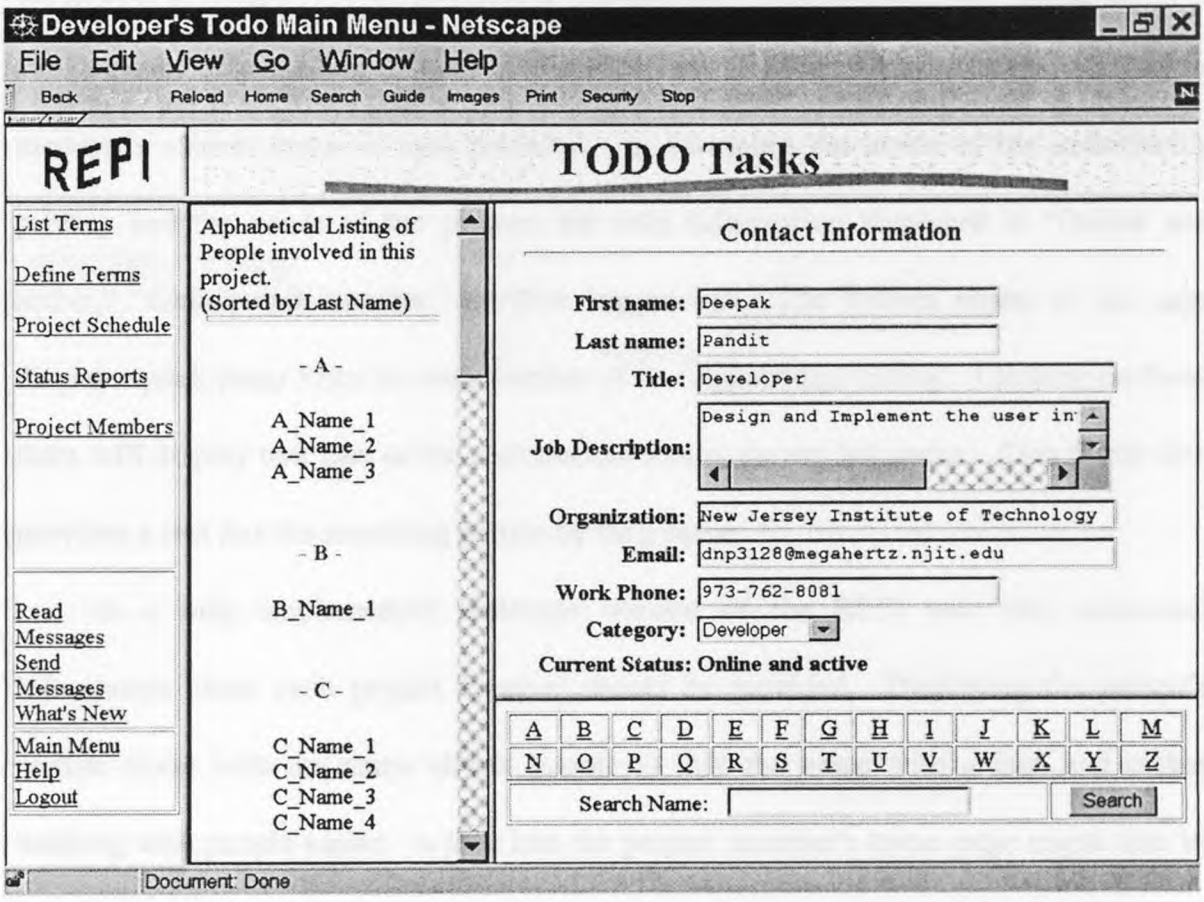


Figure 40: "Project Members" Page of the REPI Web Site

The REPI web site demo uses three frames to divide this page into meaningful and useful parts. The left frame provides an alphabetical listing of all project members, sorted by their last name. Clicking on people's last name should provide available information about that person on the top right hand side frame. This frame displays basic information such as the project member's name, job title, organization and contact information such as work phone number and email address. Within the context of the Requirements Elicitation effort, the project member's category is also displayed. In the demo version, the available categories are: "End User," "Customer," "Developer," and "Analyst." For the purposes of providing a sense of presence in a distributed environment, the project member's current status is also provided. In balancing the needs of the individual's privacy and the needs of the project, the only information displayed is "Online and active," "Online and inactive," or "Not logged in." The bottom frame of the page displays quick jump links for each section of the alphabetical listing. Clicking on these links will display that part of the alphabetical listing on the left frame. This frame also provides a text box for searching people by their name.

In a fully implemented prototype version of the REPI web site, additional information about each project member should be provided. Displaying the person's picture along with the name allows people to link the name with a face and makes working with people easier. A link into the project member's home page might also be provided for project members to get to know each other more. The options for providing a sense of presence should be expanded; perhaps with the individual's permissions more real time information about the person's on-line activities can be displayed.

3.2.9.4 User's and Developer's Help Screens

For each phase of the SEI Requirements Elicitation process model, a different help page is displayed on the REPI web site. This allows users to easily and quickly get an overview of the whole process model from the SEI framework. An on-line reference point such as this eliminates the need for users to memorize the whole process model and all the tasks for each of the phases. A simple table, containing the specific user oriented tasks and developer oriented tasks, is displayed for each of the five phases. A brief overview of the phase is also displayed above the tables in all the help screen pages. Help screen information is based on [CHRISTEL 92] and [MILLER 93]. The demo version of the web site displays a common set of screens for both client side users and development side users. More detailed and specific type of help should be provided for the users of the prototype version of the REPI web site.

Besides these help screens, additional help is provided on all menu items in all the pages of the REPI web site. As the mouse cursor moves over a menu item's link, either a text based link or a image-mapped link, the *onmouseover* and *onmouseout* events of the JavaScript event model is used to display a one line help message on the status bar of the web browser. The "ALT" attribute of the *<IMG>* tag and the *<AREA>* tag is also used to provide the same help message which is displayed near the given link. These one line help messages provide a small description of the link's destination.

3.3 REPI Web Site Evaluation

This part of the chapter evaluates the REPI web site from the perspective of its design and from the perspective of its usefulness. For the evaluation of the design, the objective is to see how easy it is for people to understand the web site. For the evaluation of its usefulness, the objective is to see how it improves on the problems of communication between developers and users. Another objective in this evaluation is to describe the advantages and disadvantages of using the web technology for Requirements Elicitation.

3.3.1 Web Site Design

This section of the thesis evaluates the design of the REPI web site. The set of criteria used here has been specifically defined for the evaluation of web sites and web based applications. The comprehension of the web pages and the overall web site is the key issue that affects the success or the failure of a web site or a web based application. Hong and Moriai have identified three areas of importance for evaluating web site design, based on the issues that affects the web site's comprehension: web structure and layout issues, navigation issues, and orientation issues [HONG 97].

3.3.1.1 Web Structure and Layout Issues

Web structure and layout issues are about the organization of information in a given web site. This area of evaluation is divided into three groups: web site structure, readability, and essential information.

Web site structure is further sub-divided into three issues: balance of web site structure, support of multiple views, and organizational metaphors. The REPI web site is judged to be well balanced; it is neither too shallow nor too deep. In fact, as shown in Figure 44, the width and the depth are almost equal for the structure of the REPI web site. The width is measured as the average number of branches or hyperlinks per page and the depth is measured as the biggest number of links from the "top" of the structure to the "bottom." This issue deals with the semantic relations among the different pages of the web site. Multiple views refer to the different ways a given web site can be explored. Organizational metaphors refer to the style used to present the overall structure of the web site. These two issues are related to the effort necessary for a visitor to construct a mental model of the web site. The REPI web site doesn't support multiple views for its pages and its choice of metaphor is not based on commonly used items. The view presented and the metaphor used is directly based on the Requirements Elicitation framework, as defined by Christel and Kang [CHRISTEL 92]. The menu pages of the web site are organized around the phases of the process model and the leaf pages of the structure are directly related to the different tasks involved in each of the five phases. The main purpose of the REPI web site is to elicit requirements using the process model defined in [CHRISTEL 92]. Any other organizational method would seem unnatural within the context of the Requirements Elicitation process model; thus providing multiple views or using a different metaphor would not increase the degree of comprehension for the web site's users.

The readability group is sub-divided into issues of document size, visual settings, and predictability. The readability of the demo version of the REPI web site can and should be improved on when the prototype version is designed. The issue of document size refers to the scrolling required to view the full information presented in a given web page. While most pages of the REPI web site do not require any scrolling, some of the most important and most used pages do require scrolling before the end of the document is reached. Some of the information presented in these "big" documents can not easily be reduced into a one screen page. Reducing the information into one screen requires breaking the full set of information into smaller chunks; thus requiring deeper level of hyperlinks before the full information can be viewed. The visual settings refers to typographical items such as background images, text color and fonts, spacing of elements, etc. These issues refer to the amount of attention demanded in a single screenful of information. Some of the forms in the REPI web site are too crowded and thus it takes an increased effort to grasp the overall information required to be entered in a given form. Some of these forms were designed to fit in one screen to reduce the scrolling needed as the users are filling out the information. This design choice should increase the usability of the forms while possibly requiring increased effort to grasp the overall "picture." As the visual settings items are partially dependent on personal preferences and matters of taste, an objective evaluation is difficult for other elements of this issue. Predictability issue refers to the labeling and annotations of hyperlinks. The contents linked by a predictable hyperlink should be directly related to the information presented by the hyperlink itself. The REPI hyperlinks are very predictable if the SEI framework is

understood by the users. The hyperlinks into the different phases are labeled with the same terms used in the framework. The hyperlinks into the different tasks are labeled with a term that is directly related to the given task's purpose. In addition to this, helpful messages are displayed on the status line and in the general area of the hyperlink itself. JavaScript is used to display the one line help message as the mouse cursor moves over the hyperlink. The "ALT" attribute of the HTML anchor tag (<A>) and the image map area tag (<AREA>) is used to provide the floating help message as the mouse cursor moves over the hyperlink. This feature of the REPI web site requires Netscape Navigator Ver. 4.0 or above.

Essential Information refers to author's identification elements and the last update date for the web page. Author's identification and other information is displayed on the "Login screen" and the "Logout screen." The last update date for the pages of the REPI doesn't apply because the fully implemented version of the web site will always display the latest information available in the requirement database. The demo version is identified as such and the latest version date is also given.

3.3.1.2 Navigation Issues

"Navigation is defined as the means by which visitors travel in the hyperspace created by a web site" [HONG 97]. Design issues in this area relate to the effort needed to fully traverse the hyperlinks of the web site. It also relates to the effort needed by the user to find the relevant information or the proper web page. This area of evaluation is divided into three groups: navigation richness, reachability, and navigation quality.

The navigation richness group is further sub-divided into four issues: search services, hyperlinks, table of contents, and navigation types. A site-wide search engine is not provided in the demo version of the REPI web site. The Requirements Elicitation process need not be followed through in a pre-defined path or even in a linear path. A search engine that allows users to find the proper phase of the process model and the specific task needed based on the type of information at hand could be very useful for people who are not very familiar with the SEI framework for Requirements Elicitation. Suppose a developer has thought of a possible problem with a requirement that was entered before. This search engine should, perhaps, refer the developer to the "Risk Assessment" task of the "Evaluation and Rationalization Phase," where such information could be entered. Without this search engine, as currently implemented on the demo version of the REPI web site, this hypothetical person could use the "Help" screens and find the proper task needed to the enter the information.

The issue of hyperlinks measures the average number of hyperlinks per web page. This issue relates to the extensiveness of hyperlinks and also relates to the overall structure of the web site, as mentioned above in the previous section. The REPI web site is discussed above with regards to this issue. The table of contents provides a clear picture of the overall structure and a menu type interface into the different pages of the web site. The design philosophy behind the REPI web site's organizational structure is also discussed above, in the previous section. A fully expanded table of contents might be of use to get an overall picture of the SEI process model. Perhaps in the prototype version, users of the web site can be given the option of a fully expanded table of contents

as a menu structure alternative to the currently implemented multilevel menu structure. But this fully expanded table of contents would need to hold as many as 45 hyperlinks to completely replace the three level menu structure currently used.

Navigation types refer to the different types of navigational support such as guided tour, indexing, navigation bars, etc.. The REPI web site provides two types of navigation types: textbased list of hyperlinks and image mapped hyperlinks. The left frame of all REPI web pages is used to list menu items that provide hyperlinks to different levels of the web site. Menu items for individual tasks provide downward links into more detailed levels of the web site. The "Main Menu" hyperlink provides upward links into the top level of the web site after a given user has logged on to the system. Phase level and task level hyperlinks provide links across the same level of the web site. The "Logout" hyperlink provides an upward link into the home page of the REPI web site, the "Login screen." The "Main Menu" screens and the phase level menu screens uses image-mapped hyperlinks to provide downward and cross sectional links, respectively.

The reachability group has issues of dead-end documents and return hyperlinks. Dead-end documents are documents that provides no hyperlink branching to any other web page within the web site. None of the web pages of the REPI web site are dead-end documents. Return hyperlinks provide history and context information. The REPI web site contains two types of return hyperlinks. The "Main Menu" hyperlink provides a link upward into the main menu for a given user. The "Logout" hyperlink provides a link back to the home page of the REPI web site, the "Login screen." Besides these two types of return links, the browser's back button can be used to traverse up the history chain.

Many cross-sectional and "same level" hyperlinks provided by the REPI web site are thought to be the most common types of links needed for users involved in the process of Requirements Elicitation. The prototype version of the REPI web site should consider providing more one level upward links, as they are lacking in the demo version of the web site. But as mentioned above, the expected usage pattern for the REPI web site would make these "Up" hyperlinks less important.

The navigation quality refers to consistency and presentation issues. Consistency issue is about the location, presentation and usage of navigational guides across the different pages of the web site. A consistent web site should provide similar types of hyperlinks at similar locations using similar presentation styles. The REPI web site's hyperlinks are always located in a consistent location. From page to page the different types of hyperlinks are consistently displayed in the exact same location. Tasks level menu links are always displayed in the top part of the left frame. "Utility" type links into the useful features of the web site are always displayed in the middle part of the left frame. Hyperlinks into different parts of the web site, such as "Logout" link and the "Help" link are always displayed in the bottom part of the left frame. Presentation issues are about the usage of images and usage of text styles when displaying hyperlinks. Consistency issue relate to the effort needed to recognize navigational patterns across the web site. A consistent user interface and consistent navigational guides help users to understand the web site easily. When hyperlinks are displayed on the right frame of the menu type pages, a similar looking image map is used consistently across all the different pages of the web site. All text-based hyperlinks consistently use the blue color to display

the hyperlink text. The left frame hyperlinks consistently use one set of fonts and font size. The right frame hyperlinks consistently use one set of fonts and font size when displayed along with an image-mapped menu.

3.3.1.3 Orientation Issues

Orientation information should address issues such as: the current location of the user within the overall structure of the web site; the navigational path used to arrive at the current location and the possible jump points from the current location into the most likely destination. Orientation information helps users "construct a mental model of the web site and map the current position to the position in the mental model" [HONG 97]. Issues involved in providing orientation information are: context, navigation history and "where to go next" information.

Orientation information should provide a context and should present the context in an easily understandable manner. The more detailed the context, the more easily it becomes to understand the web site; but detailed context information might take up valuable screen space. The REPI web site provides contextual information using many different methods. The most visible of these is the title frame used through out the REPI web site. The title frame displays the current phase of the Requirements Elicitation process model when working on one of the tasks or when one of the phase level menu is displayed. At the main menu level, the title frame clearly indicates if the web site is logged currently in the client side mode or in the development side mode. In the "utility" pages and help screens, the title frame displays the current utility function or the phase

title for the help information displayed. Besides this, the web browser's title bar displays similar contextual information. Each task page displays the current task number and the task title; other tasks in the same phase are displayed in the left frame to provide further contextual information about the current task within the given phase of the process model.

"Navigation history evaluates the extensiveness of history information built into web documents" [HONG 97]. This information should provide the user with the ability to trace backwards and to judge the current location within the overall structure of the web site. The REPI web site does not specifically provide history information in each and every page of the web site; but as described above the contextual information provided should be enough to gather the navigational history.

The "Where to go next" issue is closely linked to the type of navigational guides provided in a given page. [HONG 97] lists three different dimensions of "where to go next." The three dimensions of navigation, "move to a lower level," "move to an upper level," and "move to a page at the same level," are similar to the upward, downward, and cross sectional links described above. The REPI web site has been evaluated along these dimensions when the issues of navigational richness and reachability were discussed above, in Section 3.3.1.2.

CHAPTER 4

CONCLUSION AND FUTURE WORK

This thesis presented ideas for applying technologies of the Internet for the purposes of Requirements Elicitation. Chapter 1 of this thesis surveyed the current and near future technologies of the Internet that might be of use for Requirements Elicitation. Chapter 2 described the Software Engineering Institute's Requirements Elicitation framework and the process model. Chapter 3 of this thesis described the Requirements Elicitation Process through Internet (REPI) web site. Using the technologies described in Chapter 1 and the process model described in Chapter 2, a prototype web site was designed and implemented to explore the idea of using the Internet for the purpose of eliciting requirements. After describing the REPI web site, Chapter 3, evaluated the web site's design.

4.1 Benefits of the REPI Web Site

The process of eliciting requirements for a product to be built requires different people from different areas of expertise to work together in a group. Communication between the various members of the project is the key factor during the Requirements Elicitation part of a project because group members have to work together as a unit. Communication is necessary for information sharing which is necessary to arrive at a common understanding. Easing communications between stakeholders and developers should make the process of eliciting requirement easier, which should lead to better requirements specification and eventually a better product.

The communications between the stakeholders and the developers include both a channel, the medium for communication, and a technique, the method for communication. REPI provides assistance for both aspects of communication. The medium of communication is the World Wide Web connected via the Internet. The technique used is based on the Software Engineering Institute's framework for the Requirements Elicitation process.

Using the web as the platform, "facilitates the distribution of the application and its data to geographically-separated users on diverse computing platforms" [GIRGENSOHN 96]. One of the major benefits of REPI is that it allows people and organizations separated in space or time to exchange information and come to consensus on the needs of the people. The REPI web site provides a distributed asynchronous environment for eliciting requirements; such an environment provides several advantages as well as some limitations. Project members using the REPI web site "meet" or communicate with one another at different times, from different places. The advantages of such meetings is that "group members do not have to be physically in the same place to meet, nor must they communicate with one another at the same time" [OCKER 95]. These two characteristics of distributed asynchronous communication extend the definition of a meeting; this expanded definition of a meeting loosens the constraints in an organization and thus increases the means by which groups can accomplish their work [OCKER 95].

"Organizational and social issues have great influence on the effectiveness of communication activities" and therefore on the overall success or failure of a given project [AL-RAWAS 96]. If the communication channel is expensive between the client

side people and the development side people, limitations are placed on the number and type of communications between these two groups of people. The number and type of people selected as representatives for each side might also be limited. Sometimes surrogates or intermediaries are used as a representative to communicate with the development side people or the client side people instead of the actual clients or developers communicating with the other party; such forms of communications are labeled as indirect links. [KEIL 95] reports that direct links are better than indirect links because intermediaries might filter or distort messages between the two groups and they might not have a complete understanding of customer needs. [KEIL 95] also reports that up to a certain point the more links between customers and developers, the better it is for the development process. Another possible limitation of the expensive communication channel is that it might be restricted to one way communication [AL-RAWAS 96]. The development side people might produce documents based on their understanding and send these voluminous documents for the client side people. They might not take the time to properly validate the requirements, even if they understand the notations used in the specification document. All these limitations of the expensive communication channel reduce the accuracy of the information obtained during the Requirements Elicitation process.

The REPI web site decreases the problems associated with the above issues. Using REPI is inexpensive compared to other forms of communication channels such as meetings or passing documents. Written documents are also non-interactive communication channels. REPI provides faster turnaround time when compared with

written documents that need to be sent from one location to another. This translates into an easier form of communication because the delay is reduced between the responses. Any number of people from any location can be part of the team when using REPI for Requirements Elicitation. This should remove the limitations placed on the project team member selection. So projects can reduce or eliminate the need for intermediaries and instead use the actual stakeholders of the product regardless of their location or their number.

Another advantage of using the web is that many users are already familiar with web clients, such as Netscape Navigator or Microsoft Internet Explorer, due to the expanding growth of the Internet and the World Wide Web. As the user interface for REPI is nothing more than a series of web pages, using REPI should be as easy as browsing through the web.

Another benefit of REPI is that it imposes a structure for eliciting requirements based on the SEI's Requirements Elicitation process model. Using the REPI web site allows people involved in the process to communicate with each other more conveniently than it has been possible before.

4.2 Limitations of the REPI Web Site

The distributed asynchronous environment nature of the REPI web site creates some disadvantages, as well as the advantages mentioned above. The disadvantage of this is that many advantages of face-to-face meetings are lost in these distributed group meetings: "points of reference for indexing communication by time, place, and talk

sequence are all missing" [OCKER 95]. Video streaming technologies and other video-conferencing technologies can be used to improve on these aspects of the face-to-face meetings. Currently the "Project Members" page of the REPI web site displays the current login status of project members. This feature can be improved upon. Client side pull or server side push can be used to display the current picture of a project member's face or perhaps the current display as seen by this project member. This should provide information such as what this member is doing right now, a sense of presence that is available in a face-to-face meeting. As each contribution by any member of the project is time stamped and this information is displayed on the web pages, sequencing information is available in the demo version of the REPI web site. As the meeting takes place in a distributed environment, sequencing events by place might not apply to this type of meetings. A threaded display of contributions by subject or by person might provide better information about the talk sequence. Currently the REPI web site displays all contribution in a given category linearly sorted by time stamp.

The distributed nature of the meeting also provides greater freedom in "attending" meetings. Some members of the project might contribute their input much later than others, such that communication in these "meetings" might seem disjointed. Some level of discipline and social control needs to be created to require project members to regularly login and contribute via the REPI web site.

The disadvantages of using the web is that current generation of web browsers and the current version of the HTML are not as feature rich when compared with full blown graphical user interface platforms such as Microsoft Windows 95. So web based

applications look rather primitive compared to applications on these GUIs. Another disadvantage is that, despite the claims for cross platform portability of the web pages, the user interface looks different depending on factors such as client software and its platform, screen and color resolutions, and monitor size. Full use of the available features in the current version of the HTML is not possible if the web pages are to look as consistent as possible across web browsers and across platforms. The use of form data controls and widgets are the most visible source of inconsistent behavior. A given browser on a given GUI will always use the native data controls and widgets to display the HTML form controls; so naturally none of the HTML forms can be designed to provide the exact same "look and feel" across browsers and platforms.

The REPI web site developed for this thesis is a demo; at best it is a non-functional proof of concept prototype. In this version, JavaScript is used to enhance only some aspects of the user interface; it could be used to improve the user interface much more than it is currently done. The current version of the web site provides no error checking on user input; JavaScript can also be used for this purpose. None of the back end processes have been implemented in this demo version. Server side Java and JavaScript programming can be used along with CGI to implement a back end database used to store all the requirements information generated by the users. When certain pages of the REPI web site are loaded, the requirements database on the server needs to be queried and the proper contents of the page generated dynamically using JavaScript. The demo version only displays static web pages throughout the REPI web site. Client side Java programming, Java applets, can be used to provide a more feature rich user interface and

more front-end intelligence for many tasks of the SEI's Requirements Elicitation process model.

4.3 Future Work

The REPI web site developed for this thesis is a demo, future work on it will improve its usefulness. This section describes the three major areas in which the REPI web site can be improved upon: full implementation, user interface improvements, support for the SEI's Requirements Elicitation framework.

As mentioned before, the REPI web site is not yet fully developed. None of the back end has been implemented. For the REPI web site, a database has to be designed to store the requirements and other information generated as the project members use the web site during Requirements Elicitation. The database on the server has to be connected to the front end. Server side JavaScript or CGI programming can be used to connect the front end to the back end.

The front end for the web site has been developed using HTML and JavaScripts. The use of Java applets would allow for more front end intelligence and also provide for a richer user interface. The current user interface is designed to work consistently on many platforms and web browsers. For the next version of the REPI web site, the choice of one browser would significantly increase the possibilities for improving the user interface. Even if the current choice of cross-browser compatibility is maintained, the user interface can be improved upon by using the more recent versions of JavaScript and Dynamic HTML implementation.

Software Engineering Institute's Requirements Elicitation framework is designed to be flexible. The framework recommends using different methods and techniques based on the characteristics of the project. The process model itself can be followed in a different manner based on the project needs and the current understanding of the project goals and requirements. The type of product being developed and its history also affects the choice of techniques during the Requirements Elicitation process. The limitation with the REPI web site is that it imposes one type process model with one set of tasks implemented in one way. The REPI web site's support for Requirements Elicitation can be thought of as one instance of the SEI's process model. SEI's process model can be implemented along different lines using different methods and different path through the process model. Although the path through the process model is left up to the project members, the flexibility provided by the REPI web site doesn't match that of the Requirements Elicitation framework. This aspect of the REPI web site could be improved upon by providing alternative tasks, methods or techniques at each phase of the process model.

To summarize, the REPI web site can be improved upon by implementing the back end database connections. The user interface can be improved by either using Java applets or aggressively using Dynamic HTML, Style Sheets and JavaScripts. Increased support for the SEI's framework can be provided with alternative tasks, methods or techniques at each phase of the process model.

APPENDICES A.1: CGI EXAMPLE

```
<html>
<head>
<title>CGI SSI commands example</title>
</head>
<body>
<P>
     Current Date and Time is <!--#config timefmt="%c" -->
                              <!--#echo var="DATE_LOCAL" -->
</P>
<P>
     This page was last modified on
     <!--#flastmod file="cgi_ssi.html" --> and its current
     size is <!--#fsize file="cgi_ssi.html" -->
</P>
<!-- Include the standard signature file, below. -->
<!--#include file="signature.html" -->
</body>
</html>


<!-- This is the standard signature file, to be included in
     all other pages. -->
<P>
     Web page created by
     <A href="mailto:dnp3128@megahertz.njit.edu">
          <!--#exec cmd="graphic_signature" -->
     </A>
<center>Copyright (c) 1997</center>
</P>
```

Figure 41: CGI Example

B.1: HTML LISTS EXAMPLE

```
<html>
<head><title>HTML 3.2 <EM>Lists</EM> Example</title></head>
<body>
<ol type=I>
<lh><strong>Numbered Lists</strong>
     <li>First Item
          <ul type=disc>
               <li>Unordered list item 1
               <li>Unordered list item 2
               <li type=square>Unordered list item 3
                              (Starts using squares)
               <li>Unordered list item 4
```

```
          </ul>
     <li>Second Item
          <ol type=a>
               <li>Item number 1
                    <dl>
                         <dt>Defination List
                         <dd>A type of list which allows the
                             creation of definition
                             paragraph for the items in the
                             list.  The definition paragraph
                             is indented and displayed on
                             the next line.
                         <dt>Glossary List
                         <dd>Another name for the
                             "Definition List."  This type
                             of list can be used to list
                             dictionary entries, catalog
                             items or any other type of item
                             which requires an extensive
                             description to accompany the
                             list item.
                    </dl>
               <li>Item number 2
          </ol>
     <li>Third Item
          <ol type=1>
               <li>Item A
               <li value3>Item B (Skips an item number)
               <li>Item C
          </ol>
</ol>
</body>
</html>
```

Figure 42: HTML Lists Example

B.2: HTML TABLE EXAMPLE

```
<html>
<head><table>HTML 3.2 Table Example</table></head>
<body>
<table align=center border=10 width=100% height=100%
       cellpadding=5 cellspacing=2>
     <caption align=top>Tags and Attributes for the HTML 3.2
                        Table</caption>
     <tr align=center valign=middle
         bordercolor=blue bgcolor=white>
          <th align=center>&ltTABLE&rt Tags</th>
          <th align=center>&ltTABLE&rt Attributes</th>
```

```
          <th align=center>&ltTR&rt &amp &ltTD&rt
                              Attributes</th>
     </tr>
     <tr align=center valign=baseline>
          <td>TR</td><td>ALIGN</td><td>ALIGN</td>
     </tr>
     <tr align=center valign=baseline>
          <td>TD</td><td>BORDER</td><td>VALIGN</td>
     </tr>
     <tr align=center valign=baseline>
          <td>TH</td><td>BORDERCOLOR</td><td>COLSPAN</td>
     </tr>
     <tr align=center valign=baseline>
          <td>CAPTION</td><td>BGCOLOR</td><td>ROWSPAN</td>
     </tr>
     <tr align=center valign=baseline>

     <td> </td><td>CELLPADDING</td><td>BORDERCOLOR</td>
     </tr>
     <tr align=center valign=baseline>
          <td> </td><td>CELLSPACING</td><td>BGCOLOR</td>
     </tr>
     <tr align=center valign=baseline>
          <td> </td><td>WIDTH</td>
     </tr>
     <tr align=center valign=baseline>
          <td> </td><td>HEIGHT </td>
     </tr>
     <tr align=center valign=baseline>
     <td colspan=3>An example of a column spanning and row
                    spanning attributes</td>
     </tr>
</table>
</body>
</html>
```

Figure 43: HTML Table Example

B.3: NETSCAPE FRAME EXAMPLE

```
<-- Netscape Frame Example page: frames.htm -->
<html>
<head><title>Netscape Frame Example</title></head>
<frameset cols="1*, 4*" frameborder=yes border=5
          bordercolor=red>
     <frameset rows="10%, *" frameborder=no framespacing=10>
          <frame src="logo.htm" name="LOGO" scrolling=no
                 noresize marginwidth=0 marginheight=0>
```

```
          <frame src="nav_bar.htm name="NAV_BAR"
                 scrolling=auto marginwidth=2
                 marginheight=2>
     </frameset>
     <frameset rows="10%, *" frameborder=no framespacing=10>
          <frame src="header.htm" name="HEADER"
                 scrolling=no noresize
                 marginwidth=0 marginheight=0>
          <frame src="main.htm name="MAIN" scrolling=auto
                 marginwidth=5 marginheight=5>
     </frameset>
</frameset>
<noframes>
<body>
<P><center>This browser can not display frames.</center></P>
<ul>
<lh>Load these files to display individual frames.</lh>
     <li>
          <a href="logo.htm>
               Logo from the top left frame.
          </a>
     <li>
          <a href="header.htm>
               Header from the top right frame.
          </a>
     <li>
          <a href="nav_bar.htm>
               Navigation Bar from the bottom left frame.
          </a>
     <li>
          <a href="main.htm>
               Main contents from the bottom right frame.
          </a>
</ul>
</body>
</noframes>
</html>
```

Figure 44: Netscape Frame Example Part 1 of 3

```
<-- Netscape Frame Example page: logo.htm -->
<html>
<head><title>Logo Frame</title></head >
<body>
<img src="logo.gif">
</body>
</html>

<-- Netscape Frame Example page: header.htm -->
<html>
<head><title>Header Frame</title></head>
```

```
<body>
<center><strong>Netscape Frames Example</strong></center>
</body>
</html>

<-- Netscape Frame Example page: nav_bar.htm -->
<html>
<head><title>Navigation Bar Frame</title></head>
<body>
<ul>
     <li><a href="logo.htm target="_top">Logo Frame.</a>
     <li><a href="header.htm target="_parent">Header
                                                Frame.</a>
     <li><a href="nav_bar.htm target="MAIN">Navigation Bar
                                              Frame.</a>
     <li><a href="main.htm target="_blank">Main Frame.</a>
</ul>
</body>
</html>
```

Figure 45: Netscape Frame Example Part 2 of 3

```
<-- Netscape Frame Example page: main.htm -->
<html>
<head><title>Main Frame</title></head>
<body><center>This web page contains four frames</center>
<dl>
     <dt>Logo Frame
     <dd>Contains a logo for this web <em>site</em>.
     <dt>Header Frame
     <dd>Contains a title for this web <em>page</em>.
     <dt>Navigation Bar Frame
     <dd>Contains the navigational links for this web
         <em>site</em>.
          <ul>
               <li>Click on the "Logo" link to see the
                   effect of clearing all frames.
               <li>Click on the "Header" link to see the
                   effects of frame creation and destruction
               <li>Click on the "Navigation Bar" link to see
                   the effects of targeted frame load.
               <li>Click on the "Main" link to see the
                   effect of opening a new browser window.
          </ul>
     <dt>Main Frame
     <dd>Contains the main contents for this web
         <em>page</em>.
</dl>
</body>
</html>
```

Figure 46: Netscape Frame Example Part 3 of 3

B.4: HTML FORM EXAMPLE

```
<html>
<head><title>HTML Form Example</title></head>
<body>
<center>An example form with a text area, a select scroll
        box, a drop down box, and various types of data
        controls in it.</center>
<form method="post" action="form.cgi">
     <P>
            UserId: <input type="text" name="userid"
                           size=10><br>
          Password: <input type="password" name="password"
                           size=10>
     </P>
     <P>
           First Name: <input type="text" name="first_name"
                           size=25><br>
            Last Name: <input type="text" name="last_name"
                           size=25>
     </P>
     <P>Email Address: <input type="text" name="email"
                           size=50></P>
     <P>
          <input type="checkbox" name="checkbox_1"
                 value="Checkbox Checked" checked>Checkbox
                                                  Checked
          <input type="checkbox" name="checkbox_2"
                 value="Checkbox Cleared">Checkbox Cleared
     </P>
     <P>
          <input type="radio" name="radio" value="Radio 1">
          Radio 1
          <input type="radio" name="radio" value="Radio 2"
                 checked>Radio 2
          <input type="radio" name="radio" value="Radio 3">
          Radio 3
     </P>
     <select name="tags" multiple>
          <option value="TextArea">Text Area
          <option selected value="Select">Select
          <option value="Input">Input
     </select>
     <select name="controls" size=4 multiple>
          <option value="Text">Text
          <option selected value="Password">Password
          <option value="Checkbox">Checkbox
          <option selected value="Radio">Radio
     </select>
     Text Area: <textarea name="textarea" rows=5
                 cols=40></textarea>
     <br>
```

```
    <input type="submit" value="Submit Form">
    <input type="reset" value="Reset Form">
</form>
</body>
</html>
```

Figure 47: HTML Form Example

C.1: ESMTP EXAMPLE

```
Server>   220 megahertz.njit.edu ESMTP Sendmail 8.8.5/8.6.9
          ready at Sat, 23 Aug 1997 18:46:55 -0400 (EDT)
Client>   EHLO njit.edu
Server>   250-megahertz.njit.edu Hello megahertz.njit.edu
          [128.235.251.100], pleased to meet you
Server>   250-EXPN
Server>   250-VERB
Server>   250-8BITMIME
Server>   250-SIZE
Server>   250-DSN
Server>   250-ONEX
Server>   250-ETRN
Server>   250-XUSR
Server>   250 HELP
Client>   HELP
Server>   214-This is Sendmail version 8.8.5
Server>   214-Topics:
Server>   214- HELO EHLO MAIL RCPT DATA
Server>   214- RSET NOOP QUIT HELP VRFY
Server>   214- EXPN VERB ETRN DSN
Server>   214-For more info use "HELP <topic>".
Server>   214-To report bugs in the implementation send
              email to
Server>   214- sendmail-bugs@sendmail.org.
Server>   214-For local information send email to Postmaster
              at your site.
Server>   214 End of HELP info
Client>   QUIT
Server>   221 megahertz.njit.edu closing connection
```

Figure 48: ESMTP Example

D.1: IMAP4 STATES EXAMPLE

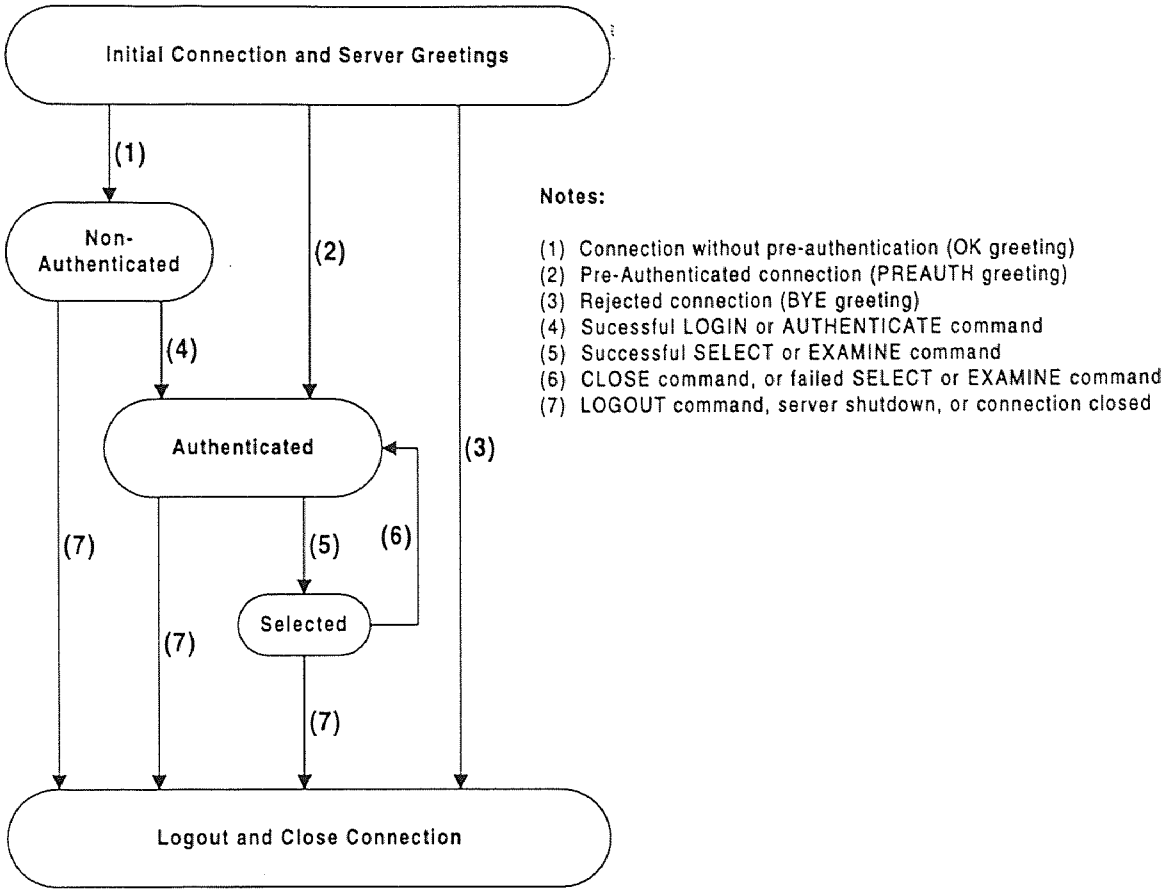


Figure 49: IMAP4 States Example. Source: Crispin, M. "Internet Message Access Protocol - Version 4rev1." Request for Comment (RFC) 2060. http://www.internic.net/rfc/rfc2060.txt. December 1996.

D.2: IMAP4rev1 EXAMPLE

```
Client>   A001 CAPABILITY
Server>   * CAPABILITY IMAP4rev1 AUTH=KERBEROS_V4
Server>   A001 OK CAPABILITY completed
Client>   A002 LOGIN dnp3128 ******
Server>   A002 OK LOGIN completed
Client>   A003 SELECT INBOX
Server>   * 172 EXISTS
Server>   * 1 RECENT
Server>   * OK [UNSEEN 12] Message 12 is first unseen
Server>   * OK [UIDVALIDITY 3857529045] UIDs valid
Server>   * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
Server>   * OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
Server>   A003 OK [READ-WRITE] SELECT completed
Client>   A004 SEARCH RECENT
Server>   * SEARCH 2 84 882
```

```
Server>   A004 OK SEARCH completed
Client>   A005 FETCH 2:4 (FLAGS BODY[HEADER.FIELDS (DATE
                          FROM)])
Server>   * 2 FETCH . . . .
Server>   * 3 FETCH . . . .
Server>   * 4 FETCH . . . .
Server>   A005 OK FETCH completed
Client>   A006 STORE 2:4 +FLAGS (\Deleted)
Server>   * 2 FETCH FLAGS (\Deleted \Seen)
Server>   * 3 FETCH FLAGS (\Deleted \Seen)
Server>   * 4 FETCH FLAGS (\Deleted \Flagged \Seen)
Server>   A006 OK STORE completed
Client>   A007 EXPUNGE
Server>   * 2 EXPUNGE
Server>   * 3 EXPUNGE
Server>   * 4 EXPUNGE
Server>   A007 OK EXPUNGE completed
Client>   A008 COPY 1 JUNK
Server>   A008 OK COPY completed
Client>   A004 LOGOUT
Server>   * BYE IMAP4rev1 Server logging out
Server>   A004 OK LOGOUT completed
```

Figure 50: IMAP4rev1 Example. Source: Crispin, M. "Internet Message Access Protocol - Version 4rev1." Request for Comment (RFC) 2060. http://www.internic.net/rfc/rfc2060.txt. December 1996.

E.1: JAVA INHERITANCE EXAMPLE

```
import java.awt.*;
import java.applet.*;

interface Shapes
 {
  abstract double getArea();
  abstract double getPerimeter();
 } /* Shapes */

class Coordinates
 {
  int x,y;

  public Coordinates (int x, int y)
   {
    this.x = x;
    this.y = y;
   } /* Coordinates */
 } /* Coordinates */

class Square extends Coordinates implements Shapes
 {
```

```
  public int width, height;

  public double getArea()
   { return (width * height); } /* getArea */
  public double getPerimeter()
   { return (2 * width + 2 * height); } /* getPerimeter */

  public Square (int x, int y, int width, int height)
   {
    super (x,y);
    this.width = width;
    this.height = height;
   } /* Square */
 } /* Square */

class Circle extends Coordinates implements Shapes
 {
  public int width, height;
  public double radius;

  public double getArea()
   { return (radius * radius * Math.PI); } /* getArea */
  public double getPerimeter()
   { return (2 * Math.PI * radius); } /* getPerimeter */

  public Circle (int x, int y, int width, int height)
   {
    super (x,y);
    this.width = width;
    this.height = height;
    radius = (double) width / 2.0;
   } /* Circle */
 } /* Circle */

public class InheritanceApplet extends Applet
 { Square box = new Square (5, 15, 25, 25);
   Circle Oval = new Circle (5, 50, 25, 25);

   public void paint (Graphics g)
    { g.drawRect (Box.x, Box.y, Box.width, Box.height);
      g.drawString ("Area: " + Box.getArea(), 50, 35);
      g.drawString ("Area: " + Box.getPerimeter(), 50, 40);
      g.drawOval (Oval.x, Oval.y, Oval.width, Oval.height);
      g.drawString ("Area: " + Oval.getArea(), 50, 70);
      g.drawString ("Area: " + Oval.getPerimeter(), 50, 75);
    } /* paint */
 } /* InheritanceApplet */
```

Figure 51: Java Inheritance Example. Source: Jamsa, Kris. *Java Now!* Jamsa Press. 1996.

E.2: JAVA THREAD EXAMPLE

```
import java.awt.*;
import java.applet.*;

public class Counter_1 extends Thread
 {
  public int value;

  public void Counter_1 ()
   { value = 0; } /* Counter_1 */
  public void count ()
   { ++value; } /* count */
  public void run ()
   {
    while (true)
     { count () } /* while */
   } /* run */
 } /* Counter_1 */

public class Counter_2 extends Thread
 {
  public int value;

  public void Counter_2 ()
   { value = 0; } /* Counter_2 */
  public void count ()
   { ++value; } /* count */
  public void run ()
   {
    while (true)
     { count () } /* while */
   } /* run */
 } /* Counter_2 */

public class ThreadApplet extends Applet implements Runnable
 {

  Font msg_one_font = new Font("TimesRoman", Font.BOLD, 18);
  FontMetrics msg_one_fontMetrics;
  String DateTime;
  Date CurrentDateTime;
  int msg_width;

  public void start ()
   {
    System.out.println ("Starting Counter 1 at " +
                        CurrentDateTime.toString());
    new Counter_1().start();
    System.out.println ("Starting Counter 1 at " +
                        CurrentDateTime.toString());
    new Counter_2().start();
```

```
  } /* start */

 public void run()
  {
   while (true)
    {
     repaint ();
     try { Counter_1.sleep (500); } /* try */
     catch (InterruptedException e) { } /* catch */
     try { Counter_2.sleep (500); } /* try */
     catch (InterruptedException e) { } /* catch */
    } /* while */
  } /* run */

 public void paint (Graphics g)
  {
   g.setFont (msg_one_font);
   msg_one_fontMetrics = g.getFontMetrics();
   DateTime = CurrentDateTime.toString();
   msg_width = (size().width -
                msg_one_fontMetrics.stringWidth(DateTime))
               / 2;
   g.drawString ("Counter 1:" + DateTime,msg_width,10);
   DateTime = CurrentDateTime.toString();
   msg_width = (size().width -
                msg_one_fontMetrics.stringWidth(DateTime))
               / 2;
   g.drawString ("Counter 2:" + DateTime,msg_width,20);
  } /* paint */
} /* ThreadApplet */
```

Figure 52: Java Thread Example

E.3: VIEW OF THE JAVA ENVIRONMENT

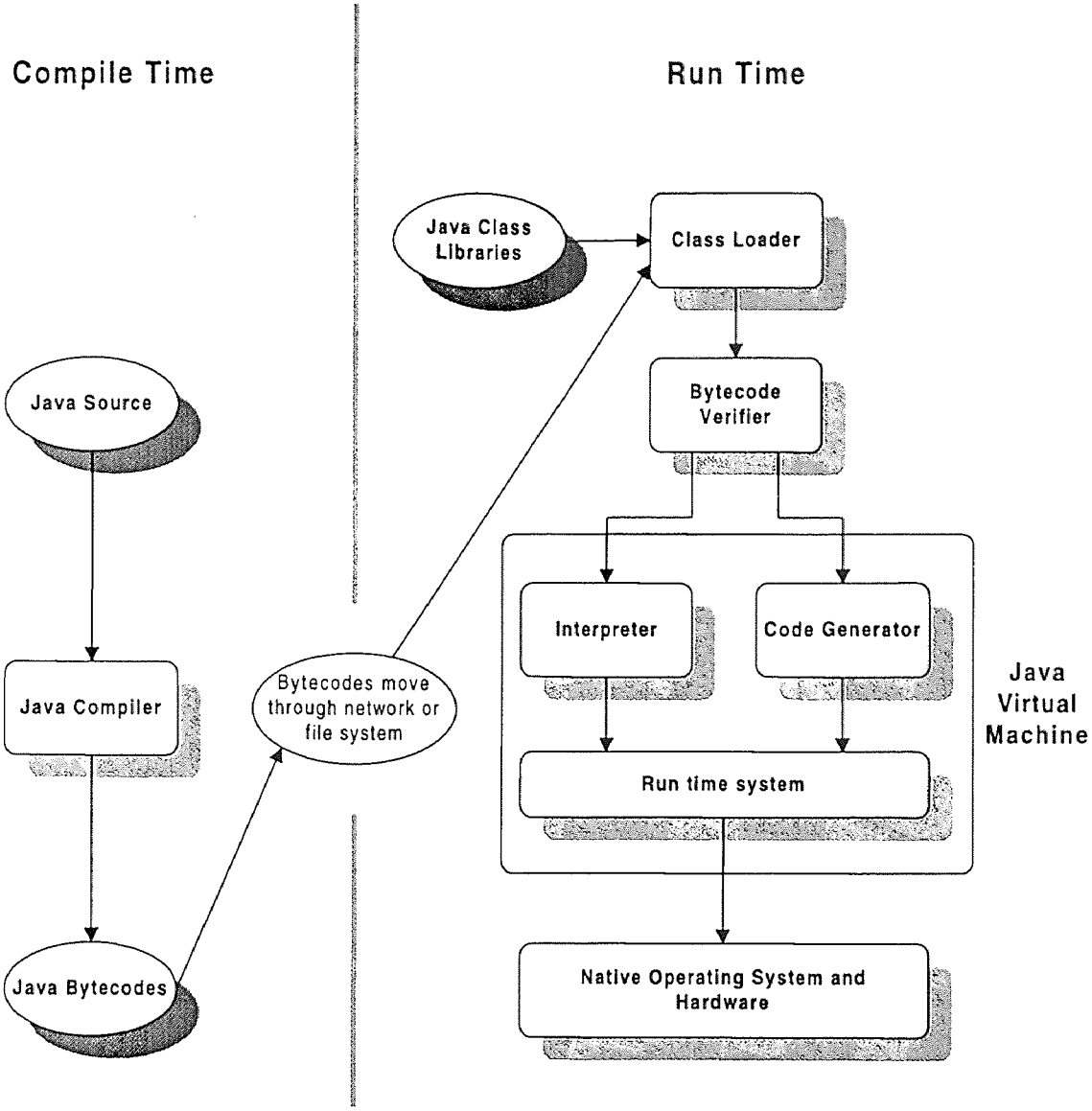


Figure 53: View of the Java Environment, Part 1 of 2. Source: Gosling, James and McGilton, Henry. "The Java Language Environment" A White Paper. Sun Microsystems, Inc. 1995. and Kramer, Douglas. *The Java Platform: A White Paper.* Sun Microsystems, Inc. 1996.

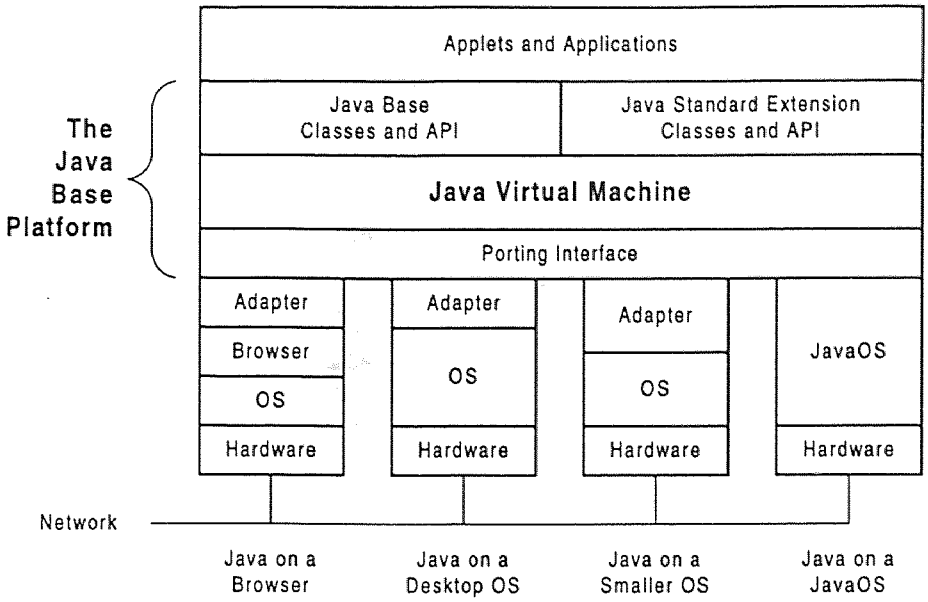


Figure 54: View of the Java Environment, Part 2 of 2. Source: Kramer, Douglas. *The Java Platform: A White Paper.* Sun Microsystems, Inc. 1996.

F.1: "Login Screen" OF THE REPI WEB SITE

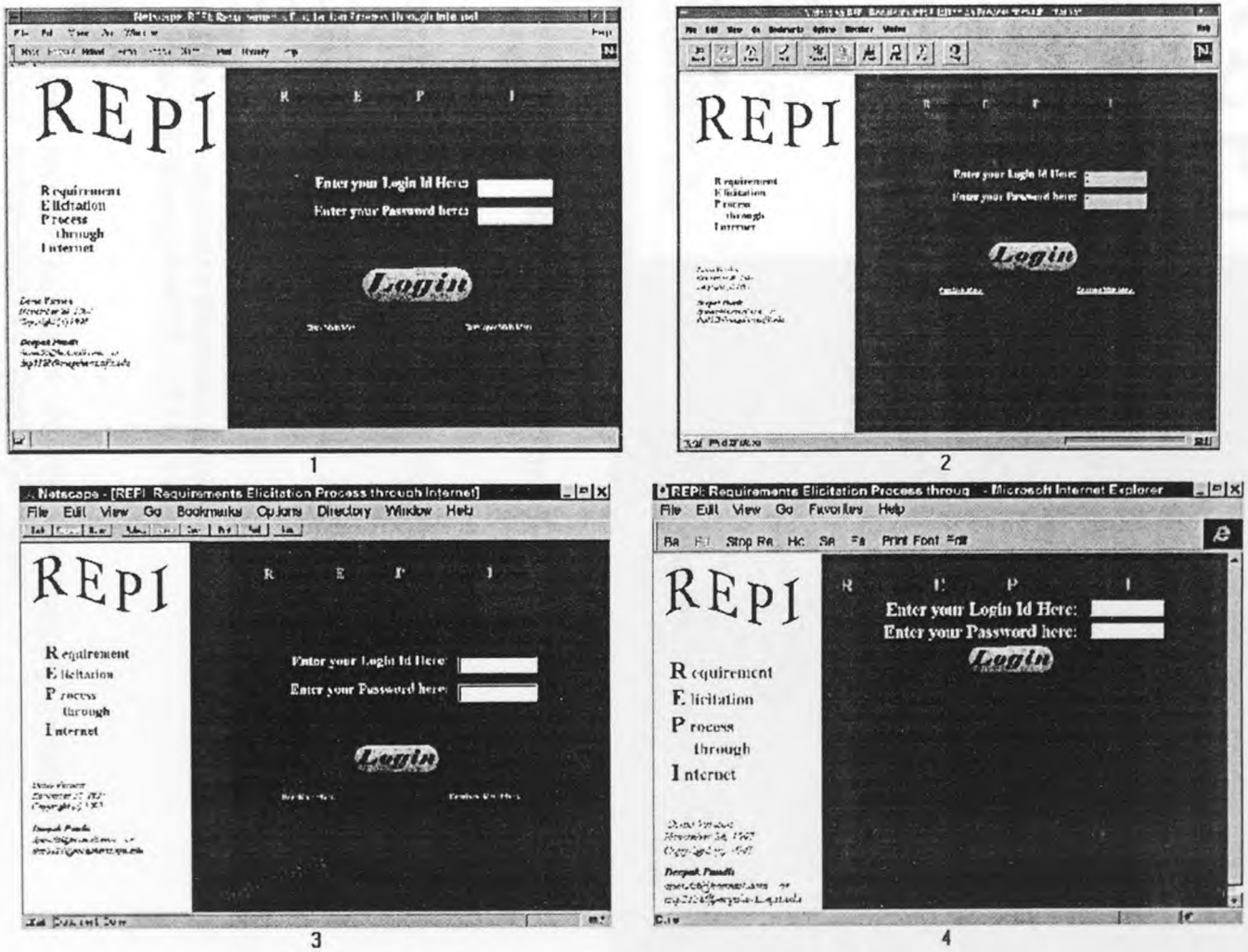


Figure 55: "Login Screen" on Various Platforms Using Different Browsers

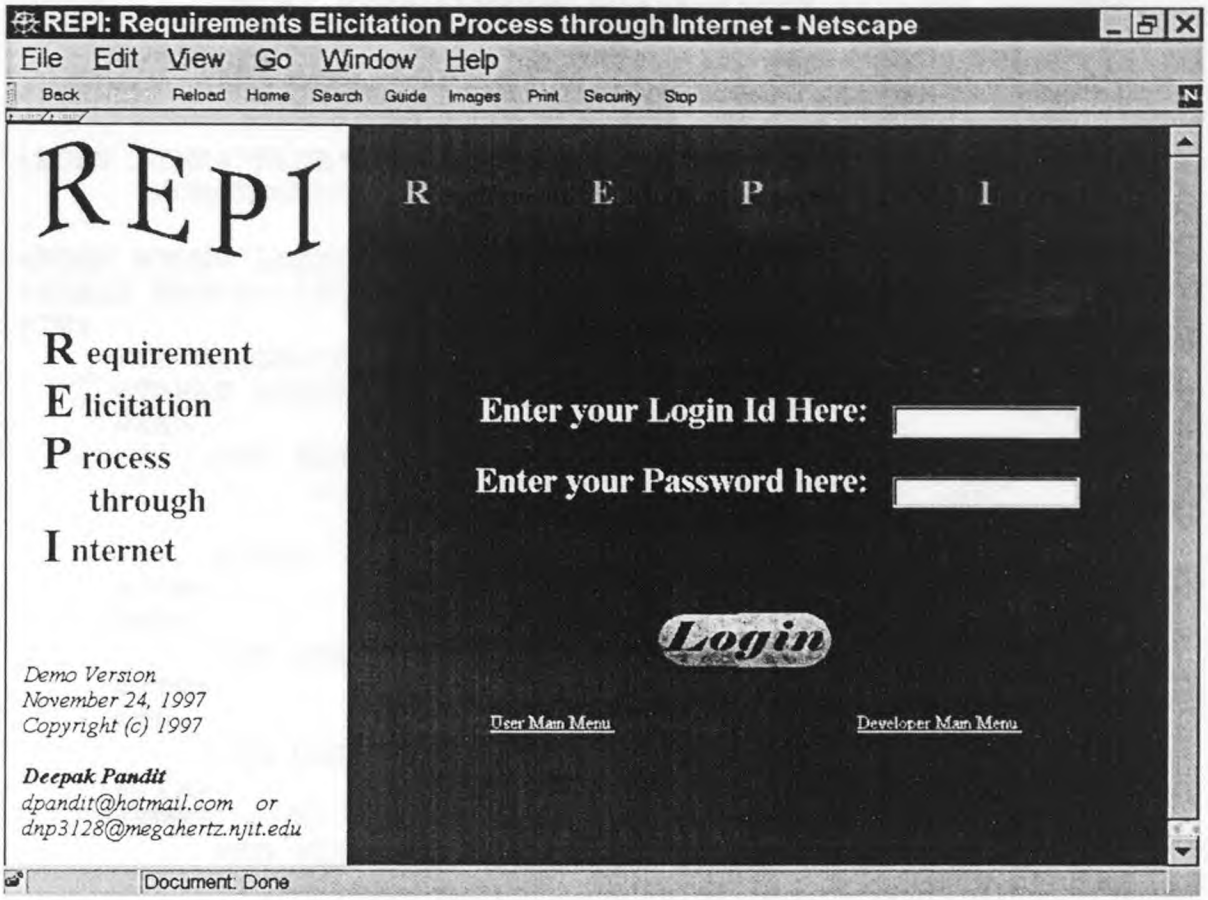


Figure 56: "Login Screen" of the REPI Web Site

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">

<!-- REPI Web Site                                          -->
<!-- Demo Version - November 24, 1997                       -->
<!-- Copyright (c) 1997                                     -->
<!--                                                        -->
<!-- Author: Deepak Pandit                                  -->
<!-- Email: dpandit@hotmail.com  or
     dnp3128@megahertz.njit.edu                             -->

<HTML>
<HEAD>
     <META NAME="Author" CONTENT="Deepak Pandit">
     <META HTTP-EQUIV="Content-Type"
          CONTENT="text/html;CHARSET=iso-8859-1">

     <SCRIPT LANGUAGE="JavaScript" SRC="REPI.JS"></SCRIPT>

     <TITLE>
```

```
            REPI: Requirements Elicitation Process through
            Internet
      </TITLE>
</HEAD>

<BODY LINK="BLUE" VLINK="BLUE" ALINK="White"
      BACKGROUND="REPI_BK1.GIF">

<FORM NAME="Login_form">
<TABLE WIDTH="100%" BORDER="0">
<TR>
      <TD ALIGN="Center" WIDTH="235">
      <TABLE WIDTH="100%" BORDER="0">
      <TR>
            <TD ALIGN="Center" WIDTH="235" COLSPAN="2">
                  <IMG SRC="REPI_ANI.GIF" WIDTH="240"
                        HEIGHT="100"><BR>
            </TD>
      </TR>
      <TR>
            <TD COLSPAN="2"> </TD>
      </TR>
      <TR>
            <TD COLSPAN="2"> </TD>
      </TR>
      <TR>
            <TD WIDTH="10"> </TD>
            <TD>
                  <STRONG><H2>
                        <FONT COLOR="Blue"><BIG>R</BIG></FONT>
                        <FONT COLOR="Red">equirement</FONT><BR>
                        <FONT COLOR="Blue"><BIG>E</BIG></FONT>
                        <FONT COLOR="Red">licitation</FONT><BR>
                        <FONT COLOR="Blue"><BIG>P</BIG></FONT>
                        <FONT COLOR="Red">rocess</FONT><BR>
                        <FONT COLOR="Red">
                                   
                              through
                        </FONT><BR>
                        <FONT COLOR="Blue"><BIG>I</BIG></FONT>
                        <FONT COLOR="Red">nternet</FONT><BR>
                  </STRONG></H2>
            </TD>
      </TR>
      <TR>
            <TD COLSPAN="2"> </TD>
      </TR>
      <TR>
            <TD COLSPAN="2"> </TD>
      </TR>
      <TR>
            <TD COLSPAN="2">
                  <ADDRESS>
                        Demo Version<BR>
```

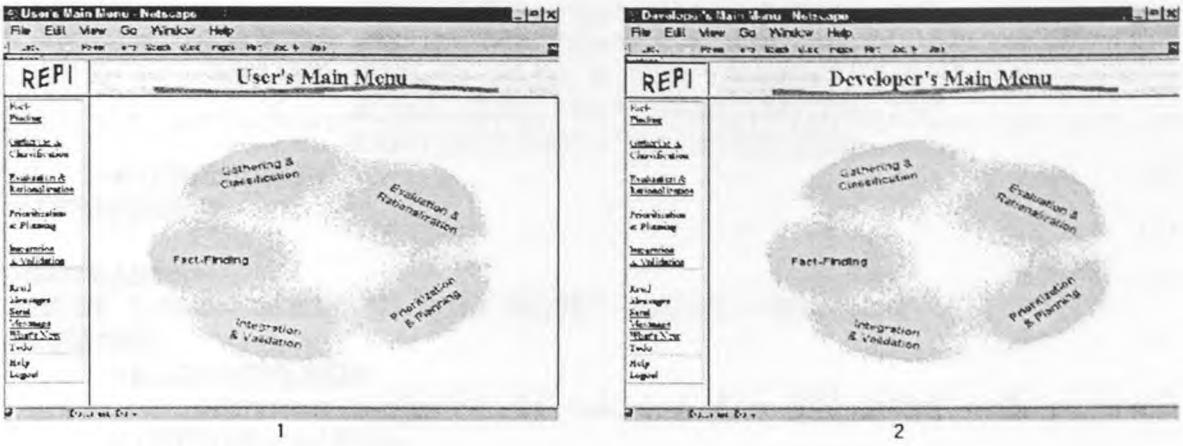
```
                    November 24, 1997<BR>
                    Copyright (c) 1997<BR>
                    <BR>
                    <STRONG>Deepak Pandit</STRONG><BR>
                    dpandit@hotmail.com
                        or<BR>
                    dnp3128@megahertz.njit.edu
               </ADDRESS>
          </TD>
     </TR>
     </TABLE>
     </TD>
     <TD WIDTH="10"> </TD>
     <TD VALIGN="TOP">
     <TABLE WIDTH="100%" BORDER="0" VALIGN="TOP">
     <TR VALIGN="Top">
          <TD ALIGN="Center" COLSPAN="2">
               <CENTER><H3><STRONG>
                    <BR>
                    <FONT COLOR="Aqua"><BIG>R</BIG></FONT>
                    <FONT COLOR="Red">equirement</FONT>
                     
                    <FONT COLOR="Aqua"><BIG>E</BIG></FONT>
                    <FONT COLOR="Red">licitation</FONT>
                     
                    <FONT COLOR="Aqua"><BIG>P</BIG></FONT>
                    <FONT COLOR="Red">rocess</FONT>
                     
                    <FONT COLOR="Red">through</FONT>
                     
                    <FONT COLOR="Aqua"><BIG>I</BIG></FONT>
                    <FONT COLOR="Red">nternet</FONT>
                    <BR><BR><BR><BR><BR>
               </STRONG></H3></CENTER>
          </TD>
     </TR>
     <TR>
          <TD COLSPAN="2"><BR><BR></TD>
     </TR>
     <TR VALIGN="Middle">
          <TD ALIGN="RIGHT" VALIGN="BOTTOM">
               <H2><STRONG>
                    <FONT COLOR="White">
                         Enter your Login Id Here: 
                    </FONT>
               </STRONG></H2>
          </TD>
          <TD><INPUT TYPE="text" NAME="Login_Id"
                    SIZE="15"></TD>
     </TR>
     <TR VALIGN="Middle">
          <TD ALIGN="RIGHT" VALIGN="BOTTOM">
               <H2><STRONG>
                    <FONT COLOR="White">
```

```
                    Enter your Password here: 
                </FONT>
            </STRONG></H2>
        </TD>
        <TD><INPUT TYPE="password" NAME="Password"
                SIZE="15"></TD>
    </TR>
    <TR>
        <TD COLSPAN="2"><BR><BR><BR></TD>
    </TR>
    <TR>
        <TD COLSPAN="2" ALIGN="Center">
            <A HREF="ERROR.HTM"
               ONMOUSEOVER="display_status('LOGIN');
               return true;"
               ONMOUSEOUT="default_status();
               return true;"
               ONFOCUS="display_status('LOGIN');
               return true;"
               ONBLUR="default_status(); return true;">
                 <IMG SRC="REPI_BT.GIF" BORDER="0"
                      WIDTH="150" HEIGHT="50">
            </A>
        </TD>
    </TR>
    <TR><TD COLSPAN="2"> </TD></TR>
    </TABLE>
    <TABLE WIDTH="100%" BORDER="0">
    <TR>
        <TD WIDTH="50%" ALIGN="CENTER">
            <A HREF="U_MAIN.HTM" TARGET="_top">
                <SMALL><FONT COLOR="White">
                    User Main Menu
                </FONT></SMALL>
            </A>

        </TD>
        <TD WIDTH="50%" ALIGN="CENTER">
            <A HREF="D_MAIN.HTM" TARGET="_top">
                <SMALL><FONT COLOR="White">
                    Developer Main Menu
                </FONT></SMALL>
            </A>
        </TD>
    </TR>
    </TABLE>
    </TD>
    <TD WIDTH="10"> </TD>
</TR></TABLE>
</FORM>
</BODY>
</HTML>
```

Figure 57: HTML Source Code for "Login Screen"

G.1: MENU PAGES OF THE REPI WEB SITE



1 2

Figure 58: User's Main Menu Screen and Developer's Main Menu Screen

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">

<!-- REPI Web Site                                          -->
<!-- Demo Version - November 24, 1997                       -->
<!-- Copyright (c) 1997                                     -->
<!--                                                        -->
<!-- Author: Deepak Pandit                                  -->
<!-- Email: dpandit@hotmail.com  or
     dnp3128@megahertz.njit.edu                             -->


<HTML>

<HEAD>
     <META NAME="Author" CONTENT="Deepak Pandit">
     <META HTTP-EQUIV="Content-Type"
CONTENT="text/html;CHARSET=iso-8859-1">

     <TITLE>User's Main Menu</TITLE>
</HEAD>

<FRAMESET ROWS="10%,90%">
     <FRAMESET COLS="15%,85%">
          <FRAME SRC="LOGO.HTM" NAME="Logo" SCROLLING="No"
                 MARGINWIDTH="0" MARGINHEIGHT="0" NORESIZE>
          <FRAME SRC="U_MAIN_T.HTM" NAME="Title"
                 SCROLLING="No" MARGINWIDTH="0"
                 MARGINHEIGHT="0" NORESIZE>
```

```
      </FRAMESET>
      <FRAMESET COLS="15%,85%">
            <FRAME SRC="U_MAIN_L.HTM" NAME="Left"
                  SCROLLING="No" MARGINWIDTH="0"
                  MARGINHEIGHT="0" NORESIZE>
            <FRAME SRC="U_MAIN_R.HTM" NAME="Right"
                  SCROLLING="No" MARGINWIDTH="5"
                  MARGINHEIGHT="5" NORESIZE>
      </FRAMESET>
</FRAMESET>

<NOFRAMES>
<BODY LINK="BLUE" VLINK="BLUE" ALINK="White">
<CENTER>
      <BIG><STRONG>
            Frames support is needed for all REPI web pages.
      </STRONG></BIG>
</CENTER>
</BODY>
</NOFRAMES>

</HTML>
```

Figure 59: HTML Source Code for "User's Main Menu"

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">

<!-- REPI Web Site                                          -->
<!-- Demo Version - November 24, 1997                       -->
<!-- Copyright (c) 1997                                     -->
<!--                                                        -->
<!-- Author: Deepak Pandit                                  -->
<!-- Email: dpandit@hotmail.com  or
      dnp3128@megahertz.njit.edu                            -->


<HTML>

<HEAD>
      <META NAME="Author" CONTENT="Deepak Pandit">
      <META HTTP-EQUIV="Content-Type"
            CONTENT="text/html;CHARSET=iso-8859-1">

      <SCRIPT LANGUAGE="JavaScript" SRC="REPI.JS"></SCRIPT>

      <TITLE>User's Main Menu Left Frame</TITLE>

</HEAD>
```

```
<BODY LINK="BLUE" VLINK="BLUE" ALINK="White"
      BACKGROUND="REPI_BK1.JPG">
<SMALL>
<TABLE BORDER="1" WIDTH="100%" BORDER="0" ALIGN="Center">
<TR VALIGN="Middle">
     <TD>
          <A HREF="U_FF.HTM" TARGET="_top"
             ALT="Examine the organization, into which, the
                  target system will be placed"
             ONMOUSEOVER="display_status('U_FF');
             return true;"
             ONMOUSEOUT="default_status(); return true;">
          Fact-<BR>Finding</A><BR>
          <BR>
          <A HREF="U_GC.HTM" TARGET="_top"
             ALT="Capture and organize the information that
                  determines what is to be built"
             ONMOUSEOVER="display_status('U_GC');
             return true;"
             ONMOUSEOUT="default_status(); return true;">
          Gathering &amp;<BR>Classification</A><BR>
          <BR>
          <A HREF="U_ER.HTM" TARGET="_top"
             ALT="Expose inconsistencies in the gathered
                  requirements and determining why the
                  information has been expressed as a
                  requirement"
             ONMOUSEOVER="display_status('U_ER');
             return true;"
             ONMOUSEOUT="default_status(); return true;">
          Evaluation &amp;<BR>Rationalization</A><BR>
          <BR>
          <A HREF="U_PP.HTM" TARGET="_top"
             ALT="Determine the relative importance of each
                  requirement"
             ONMOUSEOVER="display_status('U_PP');
             return true;"
             ONMOUSEOUT="default_status(); return true;">
          Prioritization<BR>&amp; Planning</A><BR>
          <BR>
          <A HREF="U_IV.HTM" TARGET="_top"
             ALT="Identify missing requirements and verify
                  they meet the goals"
             ONMOUSEOVER="display_status('U_IV');
             return true;"
             ONMOUSEOUT="default_status(); return true;">
          Integration<BR>&amp; Validation</A><BR>
     </TD>
</TR>
<TR VALIGN="Middle">
     <TD>
          <BR>
          <A HREF="U_READ.HTM" TARGET="_top"
             ALT="Read your new Messages"
```

```
            ONMOUSEOVER="display_status('U_READ');
            return true;"
            ONMOUSEOUT="default_status(); return true;">
         Read<BR>Messages</A><BR>
         <A HREF="U_SEND.HTM" TARGET="_top"
            ALT="Send Messages to any member of this
                 project"
            ONMOUSEOVER="display_status('U_SEND');
            return true;"
            ONMOUSEOUT="default_status(); return true;">
         Send<BR>Messages</A><BR>
         <A HREF="U_NEW.HTM" TARGET="_top"
            ALT="What's New since your last login"
            ONMOUSEOVER="display_status('U_NEW');
            return true;"
            ONMOUSEOUT="default_status(); return true;">
         What's New</A><BR>
         <A HREF="U_TODO.HTM" TARGET="_top"
            ALT="Misc items that can be useful during a
                 project"
            ONMOUSEOVER="display_status('U_TODO');
            return true;"
            ONMOUSEOUT="default_status(); return true;">
         Todo</A><BR>
      </TD>
</TR>
<TR VALIGN="Middle">
      <TD>
         <A HREF="U_HELP.HTM" TARGET="_top"
            ALT="User's Help Screen"
            ONMOUSEOVER="display_status('U_HELP');
            return true;"
            ONMOUSEOUT="default_status(); return true;">
         Help</A><BR>
         <A HREF="LOGOUT.HTM" TARGET="_top"
            ALT="Log out of the REPI Web Site"
            ONMOUSEOVER="display_status('LOGOUT');
            return true;"
            ONMOUSEOUT="default_status(); return true;">
         Logout</A><BR>
      </TD>
</TR>
</TABLE>
</SMALL>
</BODY>

</HTML>
```

Figure 60: HTML Source Code for "User's Main Menu" Left Frame

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">

<!-- REPI Web Site                                          -->
<!-- Demo Version - November 24, 1997                       -->
<!-- Copyright (c) 1997                                     -->
<!--                                                        -->
<!-- Author: Deepak Pandit                                  -->
<!-- Email: dpandit@hotmail.com  or
     dnp3128@megahertz.njit.edu                             -->


<HTML>

<HEAD>
     <META NAME="Author" CONTENT="Deepak Pandit">
     <META HTTP-EQUIV="Content-Type"
           CONTENT="text/html;CHARSET=iso-8859-1">

     <LINK REL=STYLESHEET TYPE="text/javascript"
           HREF="REPI.CSS" TITLE="Style Sheet">

     <TITLE>User's Main Menu Title Frame</TITLE>
</HEAD>

<BODY LINK="Blue" VLINK="Blue" ALINK="White"
      BACKGROUND="REPI_T.GIF">
     <CENTER><H1>
          <FONT COLOR="Red" CLASS="ScreenTitleFormat">
               User's Main Menu
          </FONT>
     </H1></CENTER>
</BODY>

</HTML>
```

Figure 61: HTML Source Code for "User's Main Menu" Title Frame

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">

<!-- REPI Web Site                                          -->
<!-- Demo Version - November 24, 1997                       -->
<!-- Copyright (c) 1997                                     -->
<!--                                                        -->
<!-- Author: Deepak Pandit                                  -->
<!-- Email: dpandit@hotmail.com  or
     dnp3128@megahertz.njit.edu                             -->

<HTML>
<HEAD>
     <META NAME="Author" CONTENT="Deepak Pandit">
```

```
      <META HTTP-EQUIV="Content-Type"
            CONTENT="text/html;CHARSET=iso-8859-1">

      <SCRIPT LANGUAGE="JavaScript" SRC="REPI.JS"></SCRIPT>

      <TITLE>User's Main Menu Right Frame</TITLE>
</HEAD>

<BODY LINK="BLUE" VLINK="BLUE" ALINK="White"
       BACKGROUND="REPI_BK2.GIF">
<BR><BR><BR>
<CENTER>
      <IMG SRC="REPI_SEI.GIF" WIDTH=652 HEIGHT=426
           ALIGN="Middle"
           BORDER="0" USEMAP="#U_SEI_MainMenu_MAP"
           ALT="SEI's Requirements Elicitation Process
                Model">
      <MAP NAME="U_SEI_MainMenu_MAP">
           <AREA SHAPE="Poly" HREF="U_IV.HTM" TARGET="_top"
                 COORDS="104,324 242,331 250,316 260,313
                         260,307 278,305 288,304 301,313
                         318,323 323,331 331,329 339,336
                         344,343 344,352 324,373 315,378
                         311,392 319,398 326,399 325,405
                         320,408 291,413 236,411 197,405
                         145,387 103,362 112,357 98,332
                         104,324"
                 ALT="Identify missing requirements and
                      verify they meet the goals"
                 ONMOUSEOVER="display_status('U_IV');
                 return true;"
                 ONMOUSEOUT="default_status();
                 return true;">
           <AREA SHAPE="Poly" HREF="U_PP.HTM" TARGET="_top"
                 COORDS="625,276 622,293 586,335 543,367
                         524,369 482,385 462,382 449,365
                         422,359 411,349 410,326 421,302
                         452,293 467,284 481,259 505,240
                         524,240 544,253 563,252 582,247
                         587,240 598,236 611,249 616,274
                         626,276 625,276"
                 ALT="Determine the relative importance of
                      each requirement"
                 ONMOUSEOVER="display_status('U_PP');
                 return true;"
                 ONMOUSEOUT="default_status();
                 return true;">
           <AREA SHAPE="Poly" HREF="U_ER.HTM" TARGET="_top"
                 COORDS="541,66 550,66 601,108 598,121
                         615,161 615,169 602,173 590,185
                         540,186 525,197 507,199 490,190
                         439,179 423,160 384,141 377,132
                         381,122 389,118 397,116 410,111
```

```
                    411,95 408,81 408,71 422,63
                    491,69 494,65 537,73 541,66"
              ALT="Expose inconsistencies in the gathered
                   requirements and determining why the
                   information has been expressed as a
                   requirement"
              ONMOUSEOVER="display_status('U_ER');
              return true;"
              ONMOUSEOUT="default_status();
              return true;">
        <AREA SHAPE="Poly" HREF="U_GC.HTM" TARGET="_top"
              COORDS="85,92 97,68 125,49 165,31
                      199,20 247,14 282,9 309,23
                      339,30 347,49 335,76 302,84
                      273,114 251,128 234,128 214,118
                      164,120 140,133 129,132 109,112
                      108,102 85,92"
              ALT="Capture and organize the information
                   that determines what is to be built"
              ONMOUSEOVER="display_status('U_GC');
              return true;"
              ONMOUSEOUT="default_status();
              return true;">
        <AREA SHAPE="Poly" HREF="U_FF.HTM" TARGET="_top"
              COORDS="81,179 117,171 157,187 195,187
                      221,217 217,237 222,252 207,261
                      187,275 151,279 132,295 110,292
                      100,282 54,277 24,259 18,227
                      14,223 14,217 22,212 26,195
                      30,189 75,186 81,179"
              ALT="Examine the organization, into which,
                   the target system will be placed"
              ONMOUSEOVER="display_status('U_FF');
              return true;"
              ONMOUSEOUT="default_status();
              return true;">
        <AREA SHAPE="DEFAULT" NOREF>
    </MAP>
</CENTER>
</BODY>
</HTML>
```

Figure 62: HTML Source Code for "User's Main Menu" Right Frame

H.1: USER'S FACT FINDING PAGES

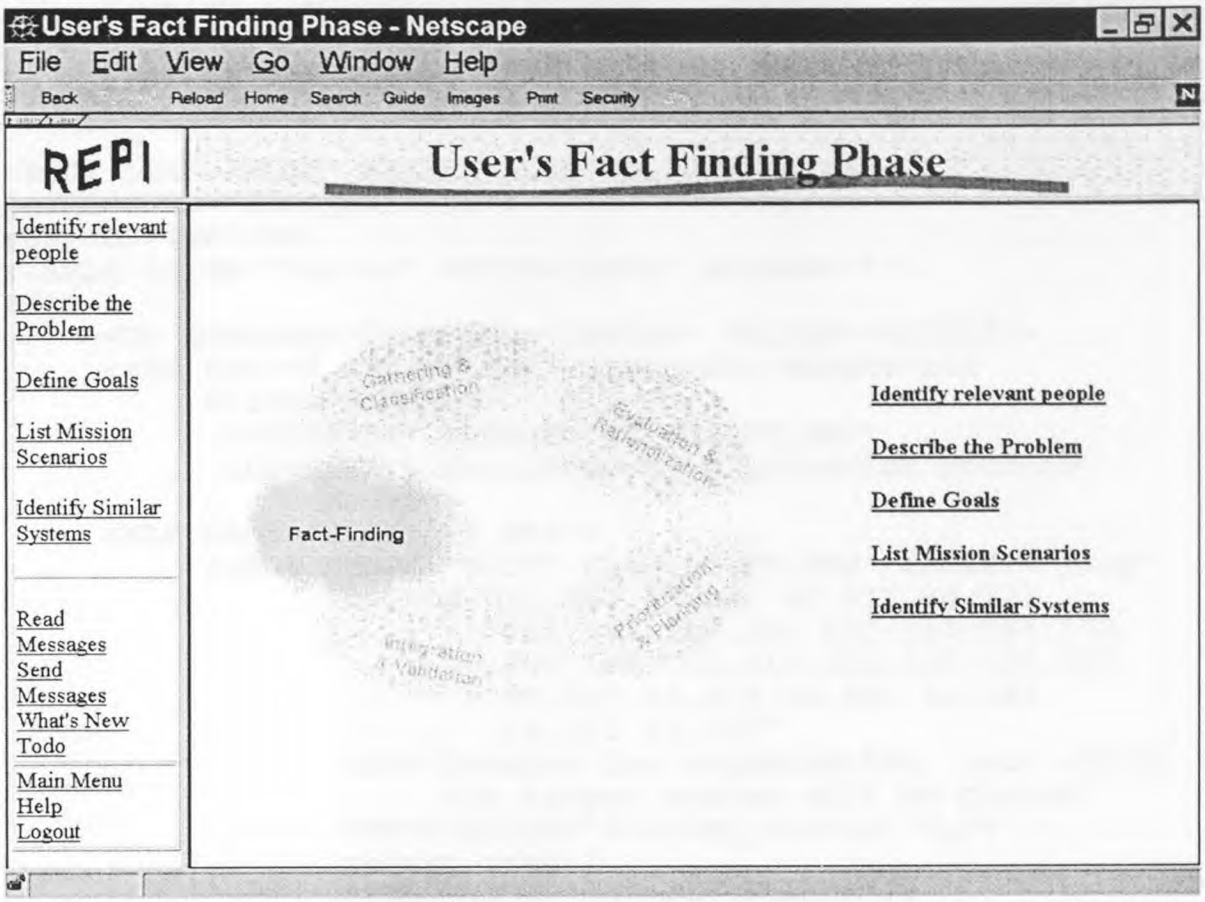


Figure 63: User's Fact Finding Menu Screen

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">

<!-- REPI Web Site                                    -->
<!-- Demo Version - November 24, 1997                 -->
<!-- Copyright (c) 1997                               -->
<!--                                                  -->
<!-- Author: Deepak Pandit                            -->
<!-- Email: dpandit@hotmail.com  or
     dnp3128@megahertz.njit.edu                       -->


<HTML>

<HEAD>
     <META NAME="Author" CONTENT="Deepak Pandit">
```

```
      <META HTTP-EQUIV="Content-Type"
            CONTENT="text/html;CHARSET=iso-8859-1">

      <SCRIPT LANGUAGE="JavaScript" SRC="REPI.JS"></SCRIPT>

      <TITLE>User's Fact Finding Right Frame</TITLE>
</HEAD>

<BODY LINK="BLUE" VLINK="BLUE" ALINK="White"
BACKGROUND="REPI_BK1.JPG">
<BR><BR><BR><BR>
<TABLE ALIGN="Center" WIDTH="100%" BORDER="0">
<TR>
      <TD ROWSPAN="2" ALIGN="CENTER" VALIGN="MIDDLE">
      <IMG SRC="R_SEI_FF.GIF" WIDTH=451 HEIGHT=340
            ALIGN="Middle"
             BORDER="0" USEMAP="#U_SEI_FF_MAP"
             ALT="SEI's Requirements Elicitation Process
                  Model">
      <MAP NAME="U_SEI_FF_MAP">
            <AREA SHAPE="POLY" HREF="U_FF.HTM" TARGET="_top"
                  COORDS="21,147 57,143 67,133 94,133
                          111,143 129,141 152,150 163,172
                          169,195 155,210 135,220 106,227
                          68,225 33,218 15,201 10,183
                          11,161 21,147"
                  ALT="Examine the organization, into which,
                       the target system will be placed"
                  ONMOUSEOVER="display_status('U_FF');
                  return true;"
                  ONMOUSEOUT="default_status();
                  return true;">
            <AREA SHAPE="POLY" HREF="U_GC.HTM" TARGET="_top"
                  COORDS="57,63 89,38 139,14 206,6
                          238,23 242,40 242,55 215,71
                          198,99 177,109 134,97 103,107
                          75,107 48,76 57,63"
                  ALT="Capture and organize the information
                       that determines what is to be built"
                  ONMOUSEOVER="display_status('U_GC');
                  return true;"
                  ONMOUSEOUT="default_status();
                  return true;">
            <AREA SHAPE="POLY" HREF="U_ER.HTM" TARGET="_top"
                  COORDS="289,50 373,57 379,50 418,88
                          415,99 428,128 420,139 408,149
                          375,149 358,161 338,149 303,141
                          293,127 263,110 265,94 286,90
                          280,55 290,50 289,50"
                  ALT="Expose inconsistencies in the gathered
                       requirements and determining why the
                       information has been expressed as a
                       requirement"
```

```
            ONMOUSEOVER="display_status('U_ER');
            return true;"
            ONMOUSEOUT="default_status();
            return true;">
      <AREA SHAPE="POLY" HREF="U_PP.HTM" TARGET="_top"
            COORDS="435,219 421,249 398,276 369,299
                    329,309 295,290 285,283 288,237
                    319,224 335,195 362,190 397,193
                    408,180 422,181 435,217 436,218
                    435,219"
            ALT="Determine the relative importance of
                 each requirement"
            ONMOUSEOVER="display_status('U_PP');
            return true;"
            ONMOUSEOUT="default_status();
            return true;">
      <AREA SHAPE="POLY" HREF="U_IV.HTM" TARGET="_top"
            COORDS="234,325 224,331 149,328 91,304
                    67,286 77,281 68,265 69,255
                    150,258 174,240 222,244 237,264
                    245,291 230,298 224,308 232,317
                    243,316 249,322 234,325"
            ALT="Identify missing requirements and
                 verify they meet the goals"
            ONMOUSEOVER="display_status('U_IV');
            return true;"
            ONMOUSEOUT="default_status();
            return true;">
      <AREA SHAPE="DEFAULT" NOREF>
</MAP>
</TD>
<TD WIDTH="5%"> </TD>
<TD ALIGN="LEFT" VALIGN="MIDDLE">
      <H4>
            <A HREF="U_FF_1.HTM" TARGET="Right"
               ALT="Identify potential stakeholders of
                    this project"
               ONMOUSEOVER="display_status('U_FF_1');
               return true;"
               ONMOUSEOUT="default_status();
               return true;">
            Identify relevant people</A><BR><BR>
      </H4>
      <H4>
            <A HREF="U_FF_2.HTM" TARGET="Right"
               ALT="Describe the problem that is to be
                    solved by this project"
               ONMOUSEOVER="display_status('U_FF_2');
               return true;"
               ONMOUSEOUT="default_status();
               return true;">
            Describe the Problem</A><BR><BR>
      </H4>
      <H4>
```

```
                <A HREF="U_FF_3.HTM" TARGET="Right"
                   ALT="List the goals to be reached by this
                        project"
                   ONMOUSEOVER="display_status('U_FF_3');
                   return true;"
                   ONMOUSEOUT="default_status();
                   return true;">
                Define Goals</A><BR><BR>
           </H4>
           <H4>
                <A HREF="U_FF_4.HTM" TARGET="Right"
                   ALT="List the general scenarios for this
                        project"
                   ONMOUSEOVER="display_status('U_FF_4');
                   return true;"
                   ONMOUSEOUT="default_status();
                   return true;">
                List Mission Scenarios</A><BR><BR>
           </H4>
           <H4>
                <A HREF="U_FF_5.HTM" TARGET="Right"
                   ALT="Identify other systems that are
                        similar to the system to be built"
                   ONMOUSEOVER="display_status('U_FF_5');
                   return true;"
                   ONMOUSEOUT="default_status();
                   return true;">
                Identify Similar Systems</A><BR><BR>
           </H4>
     </TD>
</TR>
</TABLE>
</BODY>

</HTML>
```

Figure 64: HTML Source Code for "User's Fact Finding Menu" Right Frame

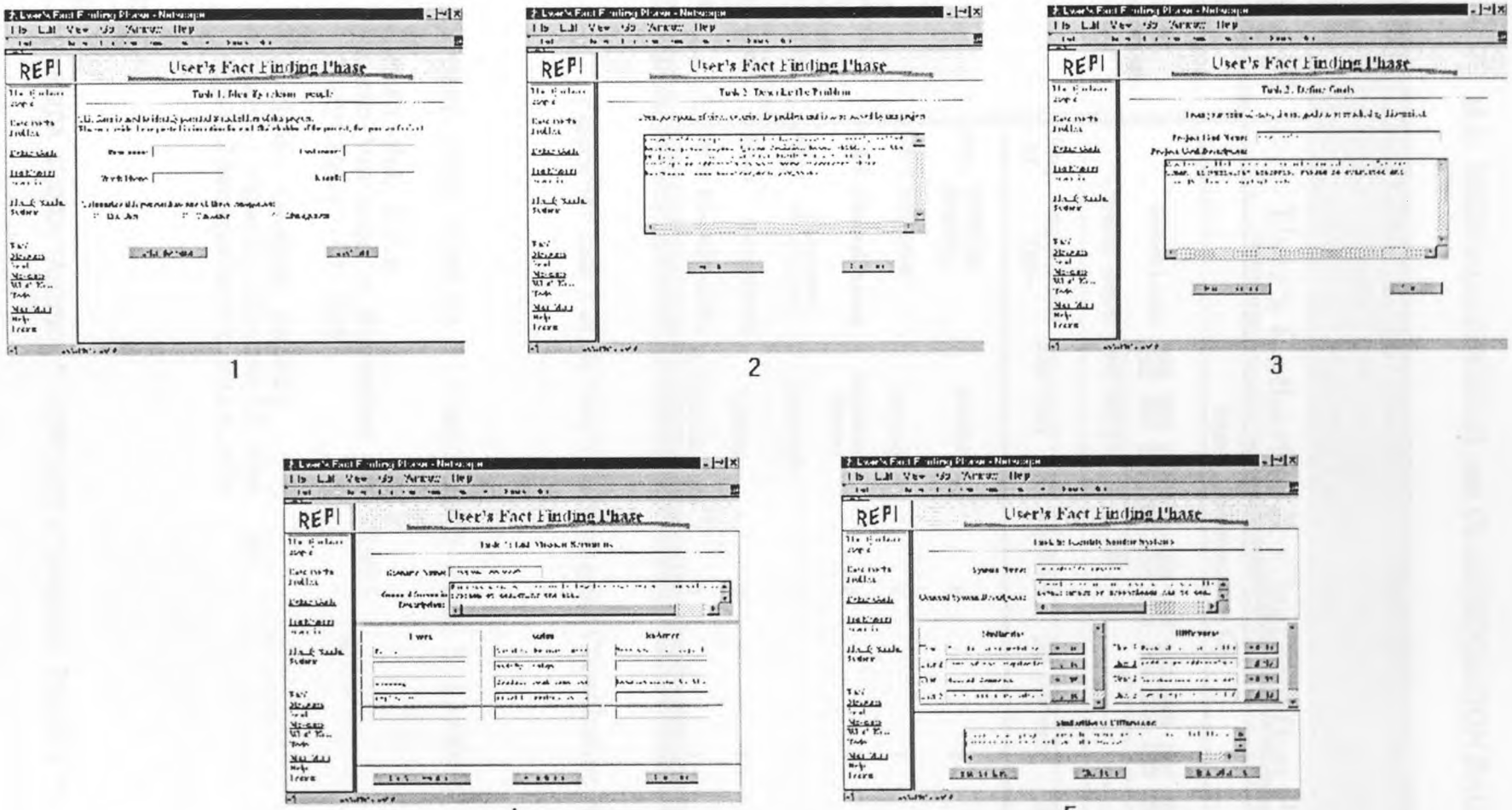


Figure 65: Five Tasks of the User's Fact Finding Phase

H.2: USER'S GATHERING and CLASSIFICATION PAGES

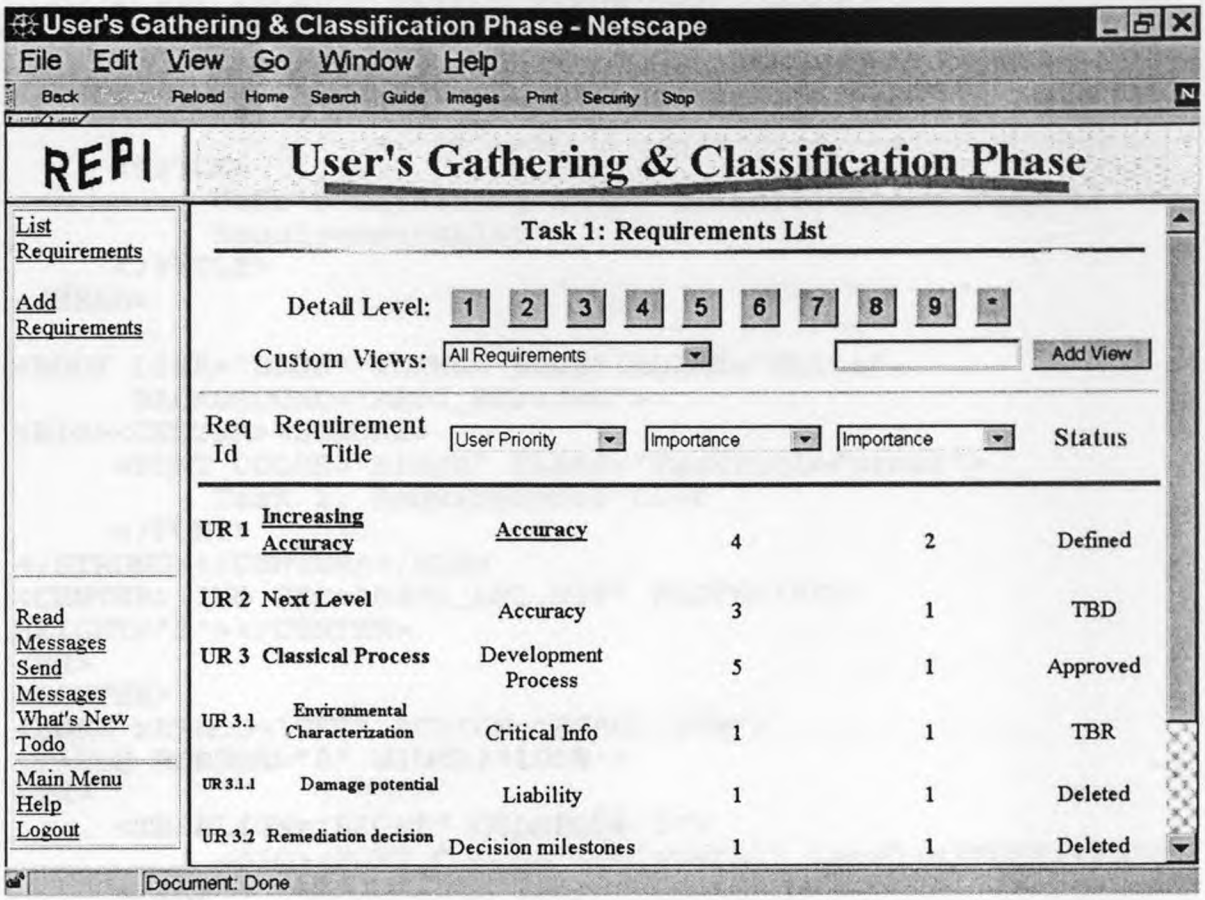


Figure 66: Task 1 of the User's Gathering and Classification Phase

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">

<!-- REPI Web Site                                          -->
<!-- Demo Version - November 24, 1997                       -->
<!-- Copyright (c) 1997                                     -->
<!--                                                        -->
<!-- Author: Deepak Pandit                                  -->
<!-- Email: dpandit@hotmail.com  or
     dnp3128@megahertz.njit.edu                             -->


<HTML>

<HEAD>
     <META NAME="Author" CONTENT="Deepak Pandit">
```

```
      <META HTTP-EQUIV="Content-Type"
            CONTENT="text/html;CHARSET=iso-8859-1">

      <LINK REL=STYLESHEET TYPE="text/javascript"
            HREF="REPI.CSS" TITLE="Style Sheet">

      <SCRIPT LANGUAGE="JavaScript" SRC="REPI.JS"></SCRIPT>

      <TITLE>
           User's Gathering &amp;amp; Classification Task 1:
           RequirementsList
      </TITLE>
</HEAD>

<BODY LINK="BLUE" VLINK="BLUE" ALINK="White"
       BACKGROUND="REPI_BK2.JPG">
<BIG><CENTER><STRONG>
      <FONT COLOR="Black" CLASS="TaskTitleFormat">
           Task 1: Requirements List
      </FONT>
</STRONG></CENTER></BIG>
<CENTER><IMG SRC="REPI_LN1.GIF" WIDTH="800"
HEIGHT="5"></CENTER>
<BR>
<CENTER>
<FORM METHOD="GET" ACTION="ERROR.HTM">
<TABLE BORDER="0" WIDTH="100%">
<TR>
      <TH ALIGN="RIGHT" COLSPAN="2">
           <BIG><FONT COLOR="Red">Detail Level:</FONT></BIG>
      </TH>
      <TD COLSPAN="3">
           <TABLE BORDER="0" WIDTH="100%">
           <TR>
                <TD ALIGN="Center">
                     <BIG>
                          <INPUT TYPE="Button"
                                 NAME="Req_Level_1"
                                 VALUE=" 1 "
                                 onClick="open_error()">
                     </BIG>
                </TD>
                <TD ALIGN="Center">
                     <BIG>
                          <INPUT TYPE="Button"
                                 NAME="Req_Level_2"
                                 VALUE=" 2 "
                                 onClick="open_error()">
                     </BIG>
                </TD>
                <TD ALIGN="Center">
                     <BIG>
                          <INPUT TYPE="Button"
                                 NAME="Req_Level_3"
```

```
                    VALUE=" 3 "
                    onClick="open_error()">
      </BIG>
</TD>
<TD ALIGN="Center">
      <BIG>
            <INPUT TYPE="Button"
                    NAME="Req_Level_4"
                    VALUE=" 4 "
                    onClick="open_error()">
      </BIG>
</TD>
<TD ALIGN="Center">
      <BIG>
            <INPUT TYPE="Button"
                    NAME="Req_Level_5"
                    VALUE=" 5 "
                    onClick="open_error()">
      </BIG>
</TD>
<TD ALIGN="Center">
      <BIG>
            <INPUT TYPE="Button"
                    NAME="Req_Level_6"
                    VALUE=" 6 "
                    onClick="open_error()">
      </BIG>
</TD>
<TD ALIGN="Center">
      <BIG>
            <INPUT TYPE="Button"
                    NAME="Req_Level_7"
                    VALUE=" 7 "
                    onClick="open_error()">
      </BIG>
</TD>
<TD ALIGN="Center">
      <BIG>
            <INPUT TYPE="Button"
                    NAME="Req_Level_8"
                    VALUE=" 8 "
                    onClick="open_error()">
      </BIG>
</TD>
<TD ALIGN="Center">
      <BIG>
            <INPUT TYPE="Button"
                    NAME="Req_Level_9"
                    VALUE=" 9 "
                    onClick="open_error()">
      </BIG>
</TD>
<TD ALIGN="Center">
      <BIG>
```

```
                              <INPUT TYPE="Button"
                                     NAME="Req_Level_*"
                                     VALUE=" * "
                                     onClick="open_error()">
                         </BIG>
                    </TD>
               </TR>
               </TABLE>
          </TD>
          <TD> </TD>
</TR>
<TR>
          <TH ALIGN="RIGHT" COLSPAN="2">
               <BIG><FONT COLOR="Red">Custom Views:</FONT><BIG>
          </TH>
          <TD COLSPAN="2">
               <SELECT NAME="Req_Views" SIZE="1">
                    <OPTION>All Requirements</OPTION>
                    <OPTION>Undefined Requirements</OPTION>
                    <OPTION>Deleted Requirements</OPTION>
                    <OPTION>User Priority Ordered</OPTION>
                    <OPTION>Funcational Requirements</OPTION>
               </SELECT>
          </TD>
          <TD ALIGN="CENTER">
               <INPUT TYPE="Text" NAME="Add_View" SIZE="15">
          </TD>
          <TD ALIGN="LEFT">
               <INPUT TYPE="Button" NAME="Add_View"
                      VALUE="Add View">
          </TD>
</TR>
<TR><TD COLSPAN="6"><HR></TD></TR>
<TR>
          <TH ALIGN="Center" VALIGN="Middle">
               <BIG><FONT COLOR="Red">Req Id</FONT></BIG>
          </TH>
          <TH ALIGN="Center" VALIGN="Middle">
               <BIG><FONT COLOR="Red">
                    Requirement Title
               </FONT></BIG>
          </TH>
          <TH ALIGN="Center" VALIGN="Middle">
               <SELECT NAME="List_Col_3" SIZE="1">
                    <OPTION SELECTED>Category</OPTION>
                    <OPTION>Compliance</OPTION>
                    <OPTION>Cost</OPTION>
                    <OPTION>Dependancy</OPTION>
                    <OPTION>D Priority</OPTION>
                    <OPTION>Importance</OPTION>
                    <OPTION>Req Type</OPTION>
                    <OPTION>Status</OPTION>
                    <OPTION>Understanding</OPTION>
                    <OPTION>User Priority</OPTION>
```

```
                <OPTION>Verified By</OPTION>
            </SELECT>
        </TH>
        <TH ALIGN="Center" VALIGN="Middle">
            <SELECT NAME="List_Col_4" SIZE="1">
                <OPTION>Category</OPTION>
                <OPTION>Compliance</OPTION>
                <OPTION>Cost</OPTION>
                <OPTION>Dependancy</OPTION>
                <OPTION>D Priority</OPTION>
                <OPTION>Importance</OPTION>
                <OPTION>Req Type</OPTION>
                <OPTION>Status</OPTION>
                <OPTION>Understanding</OPTION>
                <OPTION SELECTED>User Priority</OPTION>
                <OPTION>Verified By</OPTION>
            </SELECT>
        </TH>
        <TH ALIGN="Center" VALIGN="Middle">
            <SELECT NAME="List_Col_5" SIZE="1">
                <OPTION>Category</OPTION>
                <OPTION>Compliance</OPTION>
                <OPTION>Cost</OPTION>
                <OPTION>Dependancy</OPTION>
                <OPTION>D Priority</OPTION>
                <OPTION SELECTED>Importance</OPTION>
                <OPTION>Req Type</OPTION>
                <OPTION>Status</OPTION>
                <OPTION>Understanding</OPTION>
                <OPTION>User Priority</OPTION>
                <OPTION>Verified By</OPTION>
            </SELECT>
        </TH>
        <TH ALIGN="Center" VALIGN="Middle">
            <BIG><FONT COLOR="Red">Status</FONT></BIG>
        </TH>
</TR>
<TR>
        <TD COLSPAN="6"><HR ALIGN="Center" SIZE="5" NOSHADE>
        </TD>
</TR>
<TR>
        <TD ALIGN="Left"><H4>UR 1</H4></TD>
        <TD ALIGN="Left">
            <H4><A HREF="RINFO_1.HTM">Increasing Accuracy
            </A></H4>
        </TD>
        <TD ALIGN="Center">
            <H4><A HREF="CINFO_1.HTM">Accuracy
            </A></H4></TD>
        <TD ALIGN="Center">4</TD>
        <TD ALIGN="Center">2</TD>
        <TD ALIGN="Center">Defined</TD>
</TR>
```

```
<TR>
      <TD ALIGN="Left"><H4>UR 2</H4></TD>
      <TD ALIGN="Left"><H4>Next Level</H4></TD>
      <TD ALIGN="Center">Accuracy</TD>
      <TD ALIGN="Center">3</TD>
      <TD ALIGN="Center">1</TD>
      <TD ALIGN="Center">TBD</TD>
</TR>
<TR>
      <TD ALIGN="Left"><H4>UR 3</H4></TD>
      <TD ALIGN="Left"><H4>Classical Process</H4></TD>
      <TD ALIGN="Center">Development Process</TD>
      <TD ALIGN="Center">5</TD>
      <TD ALIGN="Center">1</TD>
      <TD ALIGN="Center">Approved</TD>
</TR>
<TR>
      <TD ALIGN="Center"><H5>UR 3.1</H5></TD>
      <TD ALIGN="Center">
           <H5>Environmental Characterization</H5>
      </TD>
      <TD ALIGN="Center">Critical Info</TD>
      <TD ALIGN="Center">1</TD>
      <TD ALIGN="Center">1</TD>
      <TD ALIGN="Center">TBR</TD>
</TR>
<TR>
      <TD ALIGN="RIGHT"><H6>UR 3.1.1</H6></TD>
      <TD ALIGN="RIGHT"><H5>Damage potential</H5></TD>
      <TD ALIGN="Center">Liability</TD>
      <TD ALIGN="Center">1</TD>
      <TD ALIGN="Center">1</TD>
      <TD ALIGN="Center">Deleted</TD>
</TR>
<TR>
      <TD ALIGN="Center"><H5>UR 3.2</H5></TD>
      <TD ALIGN="Center"><H5>Remediation decision</H5></TD>
      <TD ALIGN="Center">Decision milestones</TD>
      <TD ALIGN="Center">1</TD>
      <TD ALIGN="Center">1</TD>
      <TD ALIGN="Center">Deleted</TD>
</TR>
<TR>
      <TD ALIGN="Left"><H4>UR 4</H4></TD>
      <TD ALIGN="Left"><H4>Next Step</H4></TD>
      <TD ALIGN="Center">Decision milestones</TD>
      <TD ALIGN="Center">1</TD>
      <TD ALIGN="Center">1</TD>
      <TD ALIGN="Center">Verified</TD>
</TR>
<TR>
      <TD ALIGN="Center"><H5>UR 4.1</H5></TD>
      <TD ALIGN="Center"><H5>Possible Steps</H5></TD>
      <TD ALIGN="Center">Decision milestones</TD>
```

```
      <TD ALIGN="Center">1</TD>
      <TD ALIGN="Center">1</TD>
      <TD ALIGN="Center">Deleted</TD>
</TR>
</TABLE>
</FORM>
</CENTER>
</BODY>

</HTML>
```

Figure 67: HTML Source Code for Task 1 of User's Gathering and Classification Phase

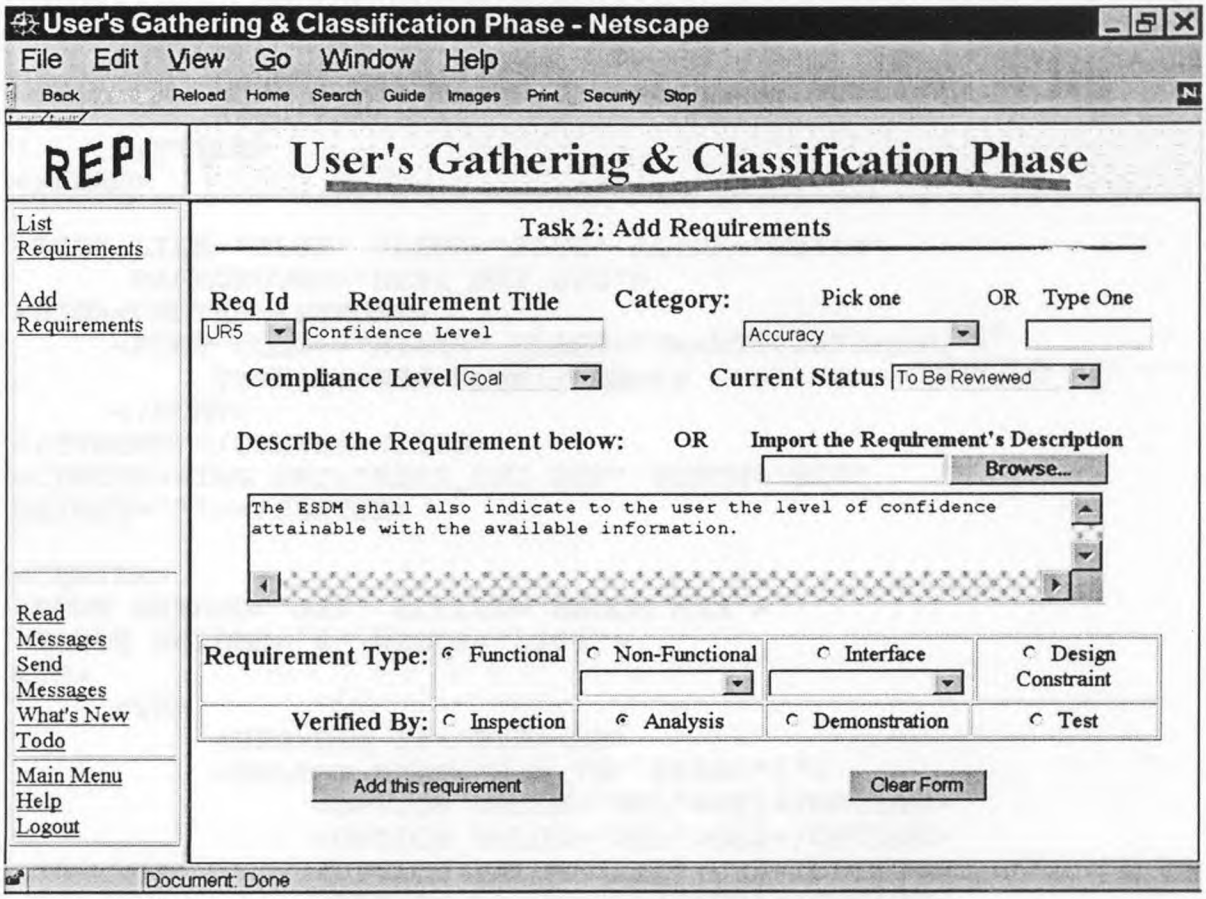


Figure 68: Task 2 of the User's Gathering and Classification Phase

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">

<!-- REPI Web Site                                    -->
<!-- Demo Version - November 24, 1997                 -->
<!-- Copyright (c) 1997                               -->
```

```
<!--                                                              -->
<!-- Author: Deepak Pandit                                        -->
<!-- Email: dpandit@hotmail.com  or
     dnp3128@megahertz.njit.edu                                   -->


<HTML>

<HEAD>
     <META NAME="Author" CONTENT="Deepak Pandit">
     <META HTTP-EQUIV="Content-Type"
          CONTENT="text/html;CHARSET=iso-8859-1">

     <LINK REL=STYLESHEET TYPE="text/javascript"
          HREF="REPI.CSS" TITLE="Style Sheet">

     <TITLE>
          User's Gathering & Classification Task 2: Add
          Requirements
     </TITLE>
</HEAD>

<BODY LINK="BLUE" VLINK="BLUE" ALINK="White"
      BACKGROUND="REPI_BK2.JPG">
<BIG><CENTER><STRONG>
     <FONT COLOR="Black" CLASS="TaskTitleFormat">
          Task 2: Add Requirements
     </FONT>
</STRONG></CENTER></BIG>
<CENTER><IMG SRC="REPI_LN1.GIF" WIDTH="800"
HEIGHT="5"></CENTER>

<CENTER>
<FORM METHOD="GET" ACTION="ERROR.HTM">
<TABLE BORDER="0" WIDTH="100%">
<TR>
     <TH>
          <BIG>Req Id</BIG><BR>
          <SELECT NAME="Req_Id" SIZE="1">
               <OPTION VALUE="UR1">UR1</OPTION>
               <OPTION VALUE="UR2">UR2</OPTION>
               <OPTION VALUE="UR3">UR3</OPTION>
               <OPTION VALUE="UR4">UR4</OPTION>
               <OPTION SELECTED VALUE="UR5">UR5</OPTION>
               <OPTION VALUE="UR6">UR6</OPTION>
               <OPTION VALUE="UR7">UR7</OPTION>
          </SELECT>
     </TH>
     <TH>
          <BIG>Requirement Title</BIG><BR>
          <INPUT TYPE="Text" NAME="Req_Title" SIZE="25"
                 VALUE="Confidence Level">
     </TH>
     <TD>
```

```
            <TABLE BORDER="0" WIDTH="100%">
            <TR>
                  <TH><BIG>Category: </BIG></TH>
                  <TD ALIGN="Center">Pick one</TD>
                  <TD ALIGN="Center">OR</TD>
                  <TD ALIGN="Center">Type One</TD>
            </TR>
            <TR>
                  <TH> </TH>
                  <TH>
                        <SELECT NAME="Req_Category" SIZE="1">
                              <OPTION SELECTED>Accuracy</OPTION>
                              <OPTION>Development process
                              </OPTION>
                              <OPTION>Decision milestones
                              </OPTION>
                              <OPTION>Critical info needs
                              </OPTION>
                              <OPTION>Liability issues</OPTION>
                        </SELECT>
                  </TH>
                  <TH> </TH>
                  <TH>
                        <INPUT TYPE="Text"
                         NAME="Req_CategoryTitle" SIZE="10">
                  </TH>
            </TR>
            </TABLE>
      </TD>
</TR>
</TABLE>
<TABLE BORDER="0" WIDTH="100%">
<TR>
      <TH>
            <BIG>Compliance Level</BIG>
            <SELECT NAME="Req_ComplianceLevel" SIZE="1">
                  <OPTION VALUE="Mandatory">Mandatory</OPTION>
                  <OPTION SELECTED VALUE="Goal">Goal</OPTION>
                  <OPTION VALUE="Objective">Objective</OPTION>
                  <OPTION VALUE="Optional">Optional</OPTION>
            </SELECT>
      </TH>
      <TH>
            <BIG>Current Status</BIG>
            <SELECT NAME="Req_CurrentStatus" SIZE="1">
                  <OPTION VALUE="TBD">To Be Determined</OPTION>
                  <OPTION SELECTED VALUE="TBR">To Be Reviewed
                  </OPTION>
                  <OPTION VALUE="Defined">Defined</OPTION>
                  <OPTION VALUE="Approved">Approved</OPTION>
                  <OPTION VALUE="Verified">Verified</OPTION>
                  <OPTION VALUE="Deleted">Deleted</OPTION>
            </SELECT>
      </TH>
```

```
</TR>
</TABLE>
<BR>
<A NAME="Input"></A>
<TABLE BORDER="0" WIDTH="100%">
<TR>
     <TH VALIGN="Top">
          <BIG>Describe the Requirement below:</BIG>
     </TH>
     <TH VALIGN="Top">OR</TH>
     <TH>
          Import the Requirement's Description<BR>
          <INPUT TYPE="File" NAME="Req_DescriptionImport"
                 SIZE="15">
     </TH>
</TR>
<TR>
     <TH COLSPAN="3">
          <TEXTAREA NAME="Req_Description" COLS="70"
                    ROWS="3">
The ESDM shall also indicate to the user the level of
confidence attainable with the available information.
          </TEXTAREA>
     </TH>
</TR>
</TABLE>
<BR>
<TABLE BORDER="1" WIDTH="100%">
<TR>
     <TH ALIGN="right" VALIGN="Top">
          <BIG>Requirement Type:</BIG>
     </TH>
     <TD ALIGN="Center" VALIGN="Top">
          <INPUT TYPE="Radio" NAME="Req_Type"
                 VALUE="Functional" ALIGN="Middle" CHECKED>
          Functional
     </TD>
     <TD ALIGN="Center">
          <INPUT TYPE="Radio" NAME="Req_Type"
                 VALUE="NonFunctional" ALIGN="Middle">
          Non-Functional<BR>
          <SELECT NAME="Req_NonFunType" SIZE="1">
               <OPTION> </OPTION>
               <OPTION VALUE="Performance">Performance
               </OPTION>
               <OPTION VALUE="Security">Security</OPTION>
               <OPTION VALUE="Maintainability">
                    Maintainability
               </OPTION>
               <OPTION VALUE="Portability">Portability
               </OPTION>
               <OPTION VALUE="Extensiblity">Extensiblity
               </OPTION>
          </SELECT>
```

```
      </TD>
      <TD ALIGN="Center" COLSPAN="2">
            <INPUT TYPE="Radio" NAME="Req_Type"
                  VALUE="Interface" ALIGN="Middle">
            Interface<BR>
            <SELECT NAME="Req_InterfaceType" SIZE="1">
                  <OPTION> </OPTION>
                  <OPTION VALUE="User">User</OPTION>
                  <OPTION VALUE="Software">Software</OPTION>
                  <OPTION VALUE="Communications">Communications
                  </OPTION>
                  <OPTION VALUE="Hardware">Hardware</OPTION>
                  <OPTION VALUE="External">External</OPTION>
            </SELECT>
      </TD>
      <TD ALIGN="Center" VALIGN="Top">
            <INPUT TYPE="Radio" NAME="Req_Type"
                  VALUE="Design_Constraint" ALIGN="Middle">
            Design Constraint
      </TD>
</TR>
<TR>
      <TH ALIGN="right"><BIG>Verified By:</BIG></TH>
      <TD ALIGN="Center">
            <INPUT TYPE="Radio" NAME="Req_VerifyType"
                  VALUE="Inspection" ALIGN="Middle">
            Inspection
      </TD>
      <TD ALIGN="Center">
            <INPUT TYPE="Radio" NAME="Req_VerifyType"
                  VALUE="Analysis" ALIGN="Middle" CHECKED>
            Analysis
      </TD>
      <TD ALIGN="Center">
            <INPUT TYPE="Radio" NAME="Req_VerifyType"
                  VALUE="Demonstration" ALIGN="Middle">
            Demonstration
      </TD>
      <TD ALIGN="Center" COLSPAN="2">
            <INPUT TYPE="Radio" NAME="Req_VerifyType"
                  VALUE="Test" ALIGN="Middle">
            Test
      </TD>
</TR>
</TABLE>
<BR>
<TABLE BORDER="0" WIDTH="100%">
<TR>
      <TD ALIGN="Center" WIDTH="50%">
            <INPUT TYPE="Button" NAME="U_GC_2_Enter"
                  VALUE="Add this requirement"
                  onClick="open_error()">
      </TD>
      <TD ALIGN="Center" WIDTH="50%">
```

```
            <INPUT TYPE="Reset" VALUE="Clear Form">
        </TD>
</TR>
</TABLE>
</FORM>
</CENTER>
</BODY>

</HTML>
```

Figure 69: HTML Source Code for Task 2 of User's Gathering and Classification Phase

H.3: USER'S EVALUATION and RATIONALIZATION PAGES

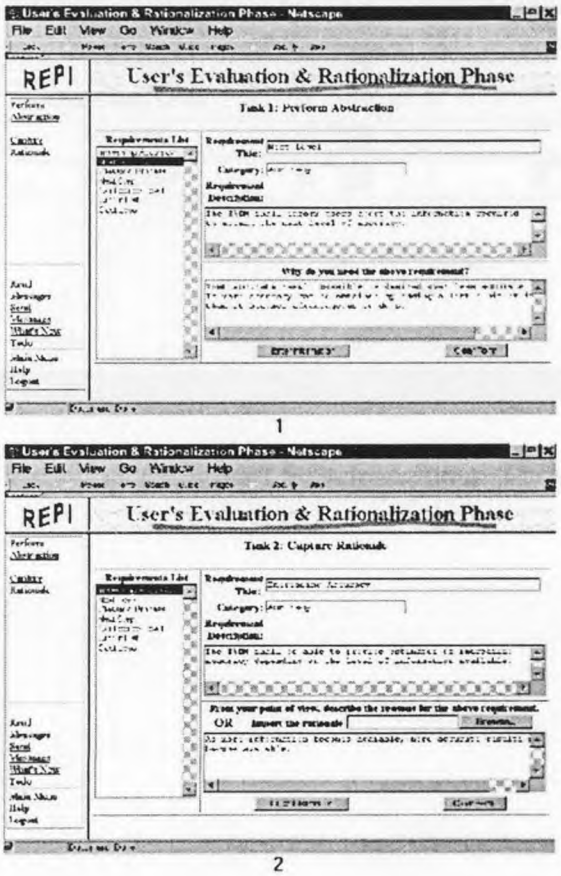


Figure 70: Two Tasks of the User's Evaluation and Rationalization Phase

H.4: USER'S PRIORITIZATION and PLANNING PAGES

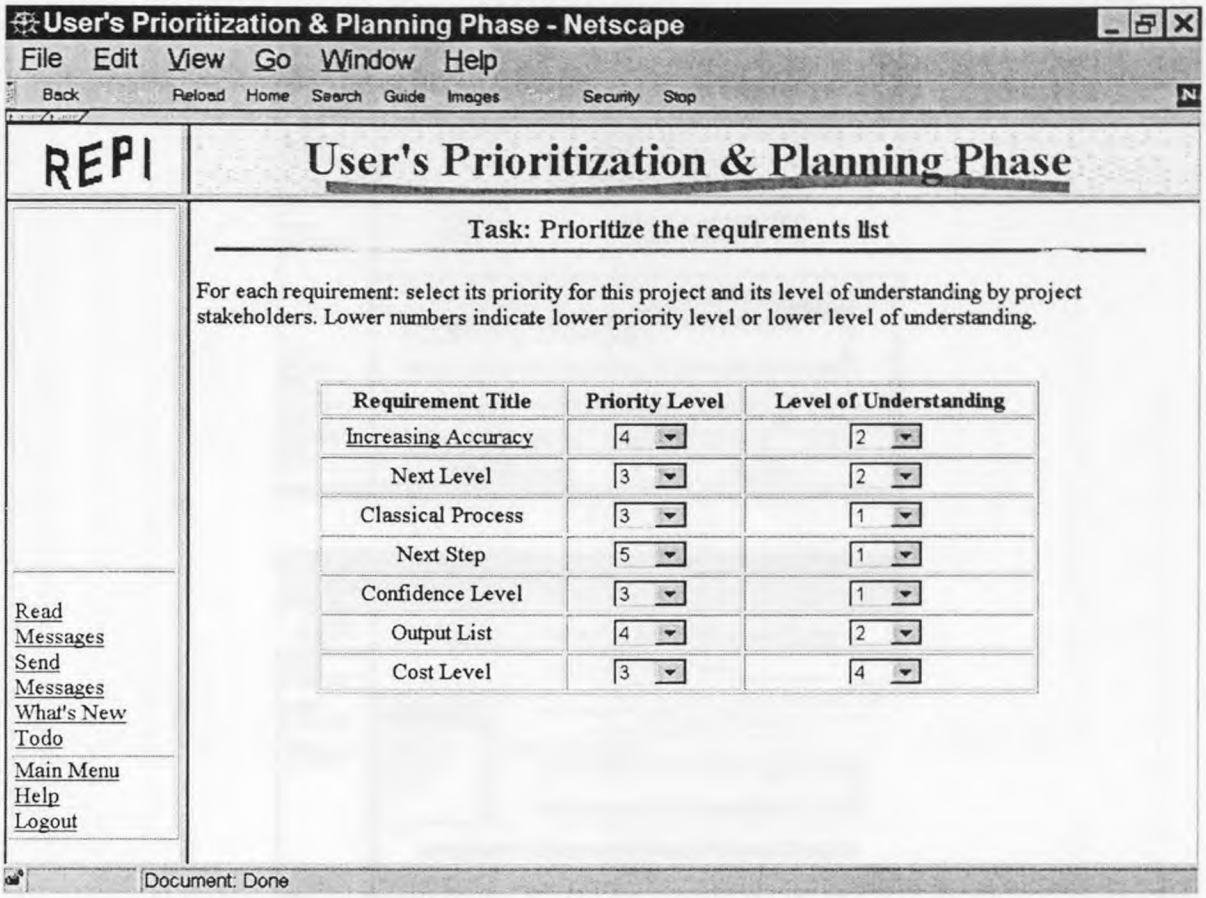


Figure 71: Task 1 of the User's Prioritization and Planning Phase

H.5: USER'S INTEGRATION and VALIDATION PAGES

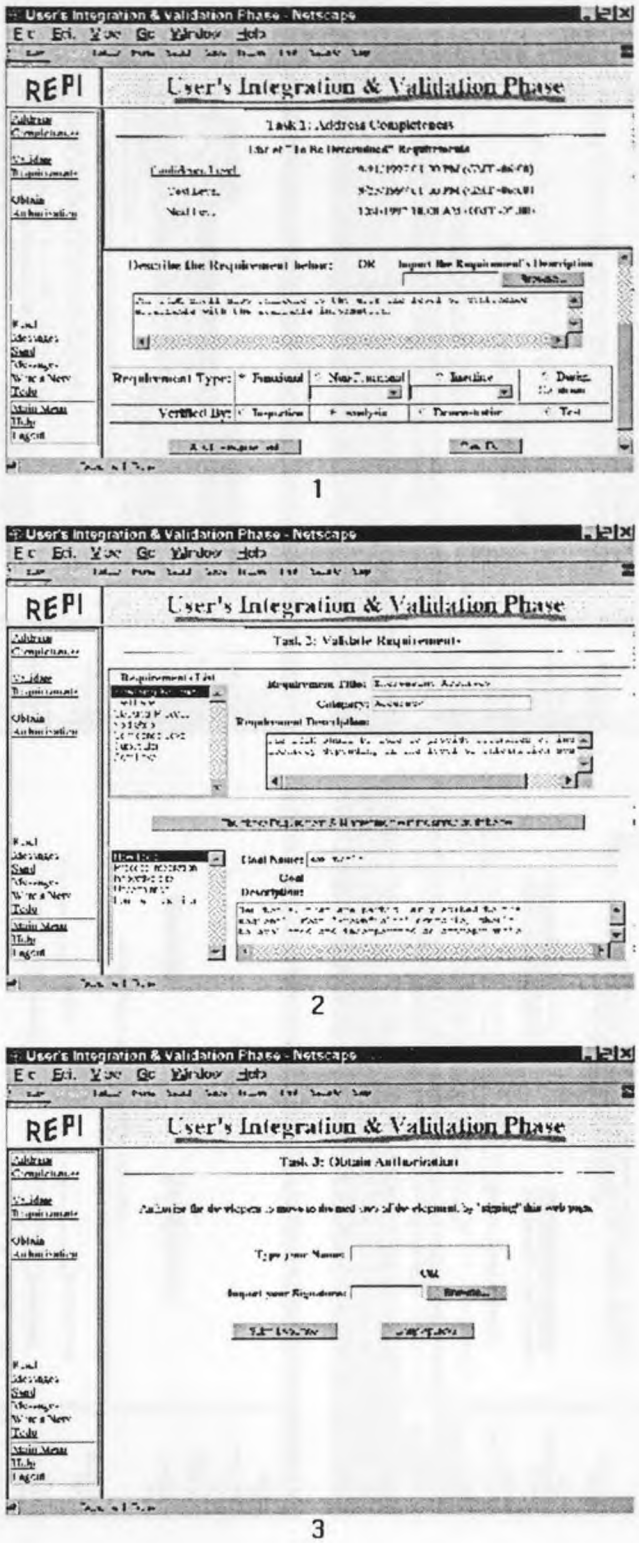


Figure 72: Three Tasks of the User's Integration and Validation Phase

APPENDIX I.1: DEVELOPER'S FACT FINDING PAGES

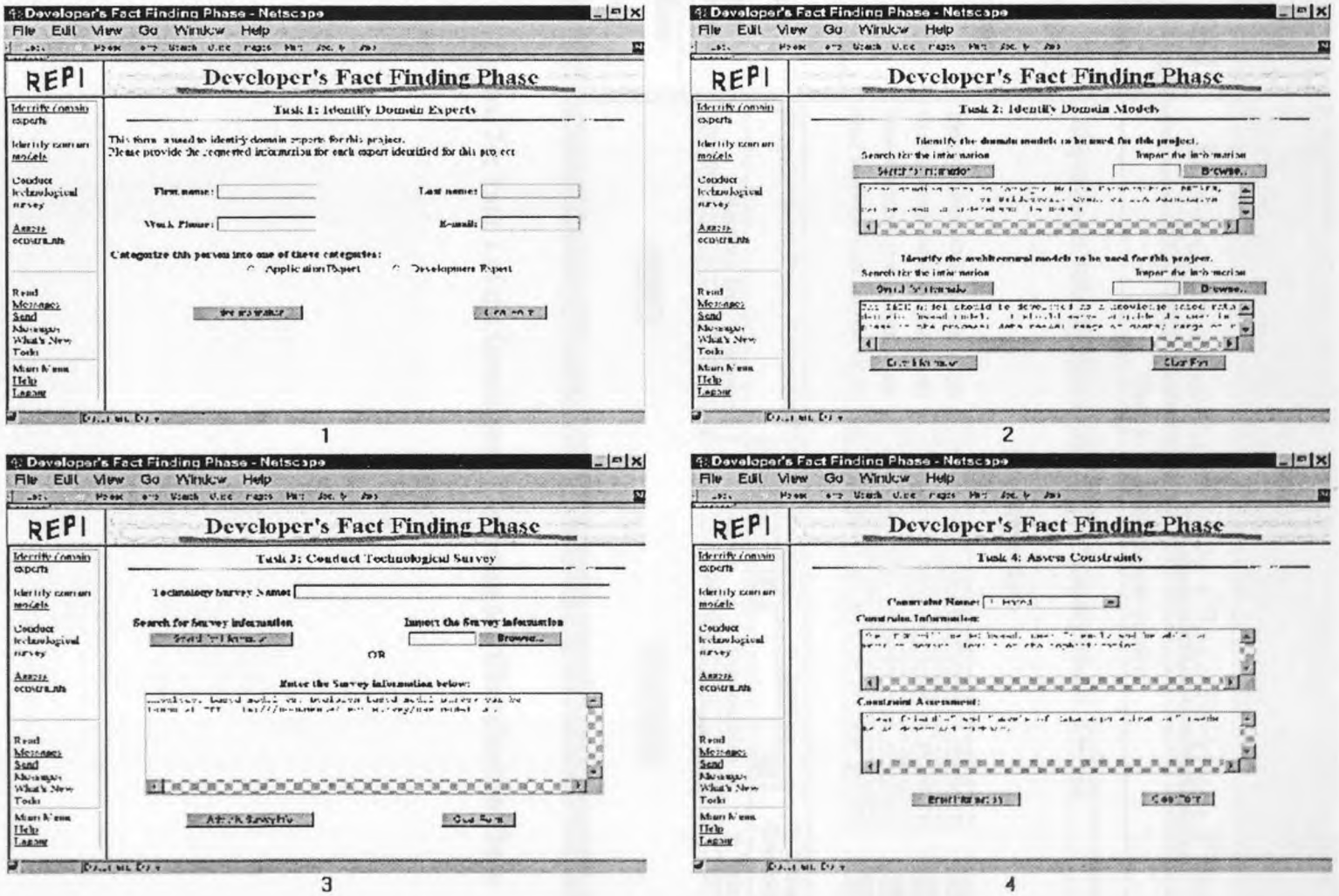


Figure 73: Four Tasks of the Developer's Fact Finding Phase

I.2: DEVELOPER'S GATHERING and CLASSIFICATION PAGES

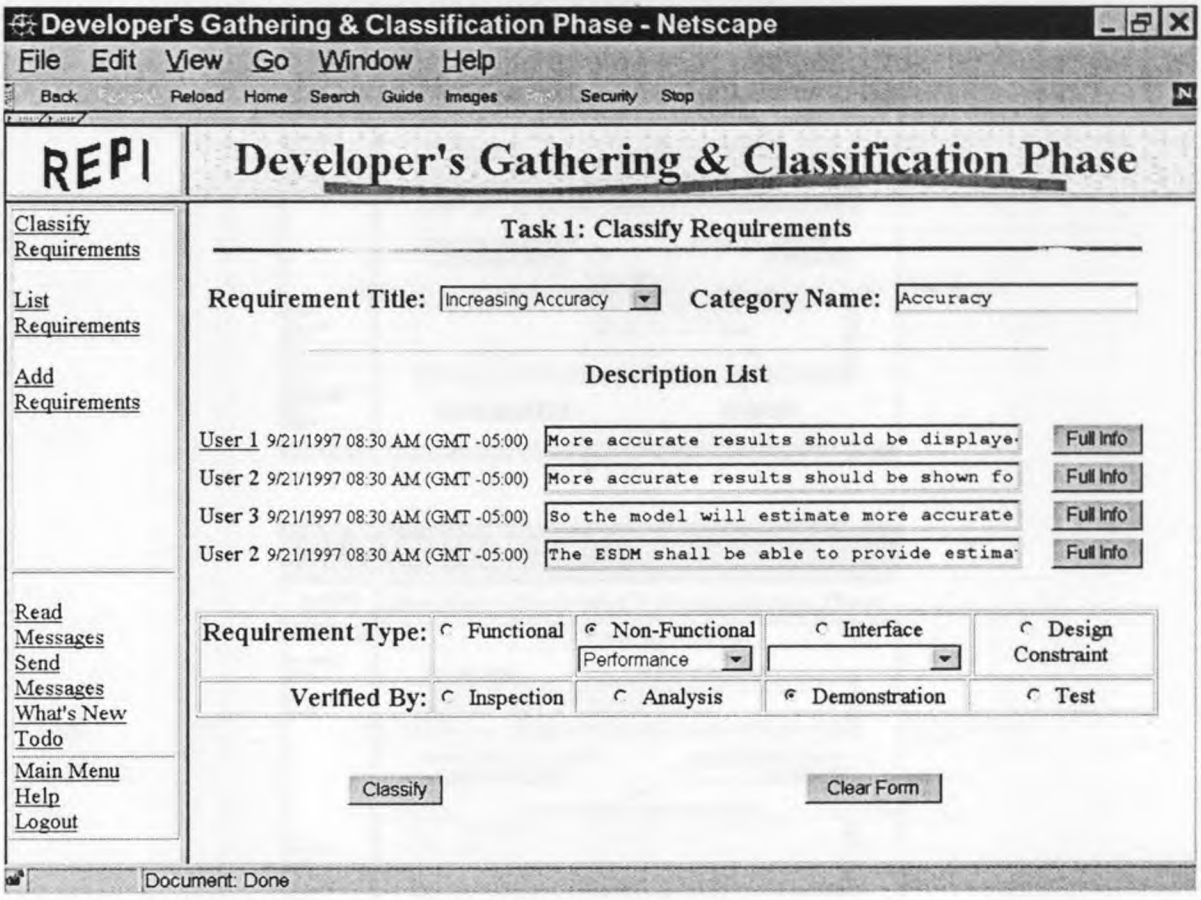
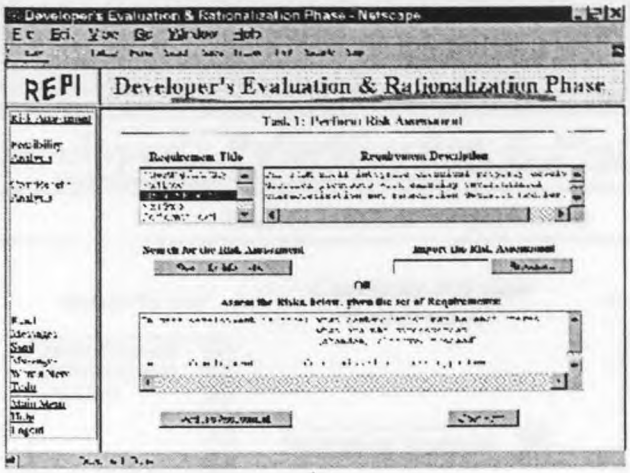
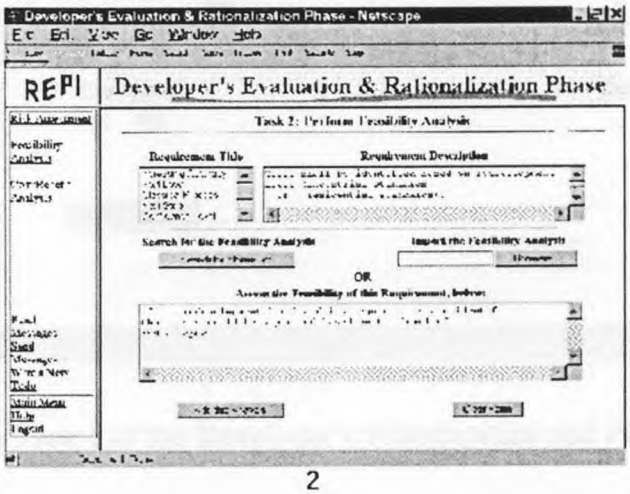


Figure 74: Task 1 of the Developer's Gathering and Classification Phase

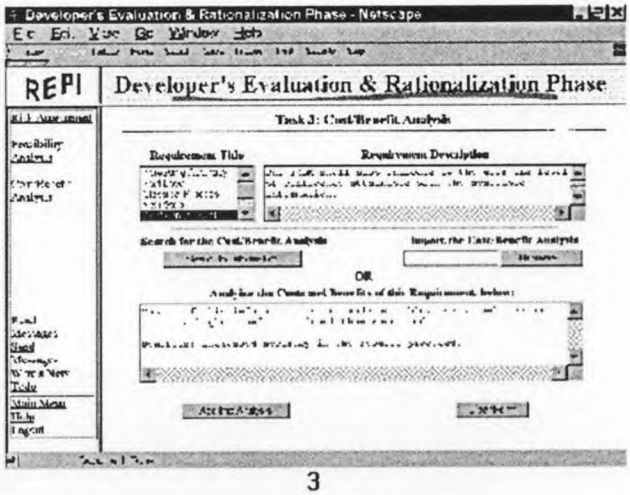
I.3: DEVELOPER'S EVALUATION and RATIONALIZATION PAGES



1



2



3

Figure 75: Three Tasks of the Developer's Evaluation and Rationalization Phase

I.4: DEVELOPER'S PRIORITIZATION and PLANNING PAGES

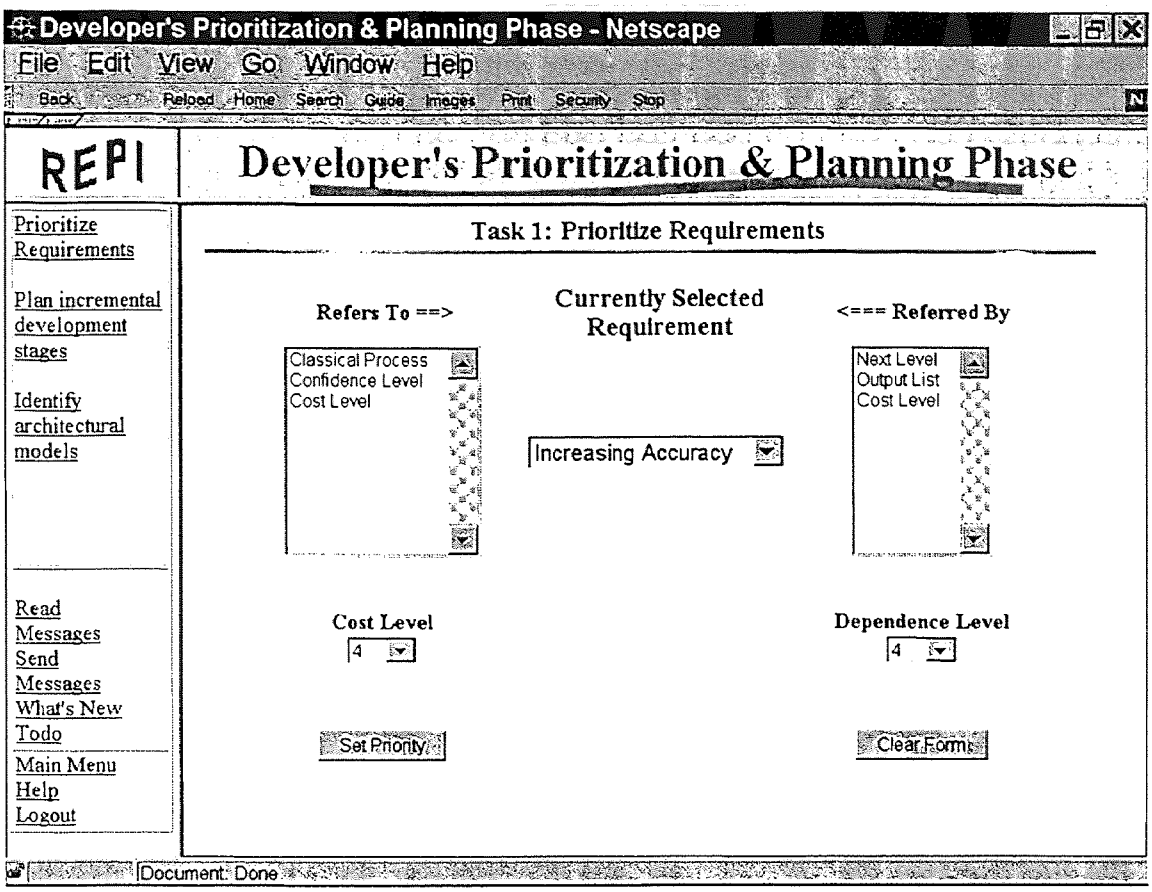


Figure 76: Task 1 of the Developer's Prioritization and Planning Phase

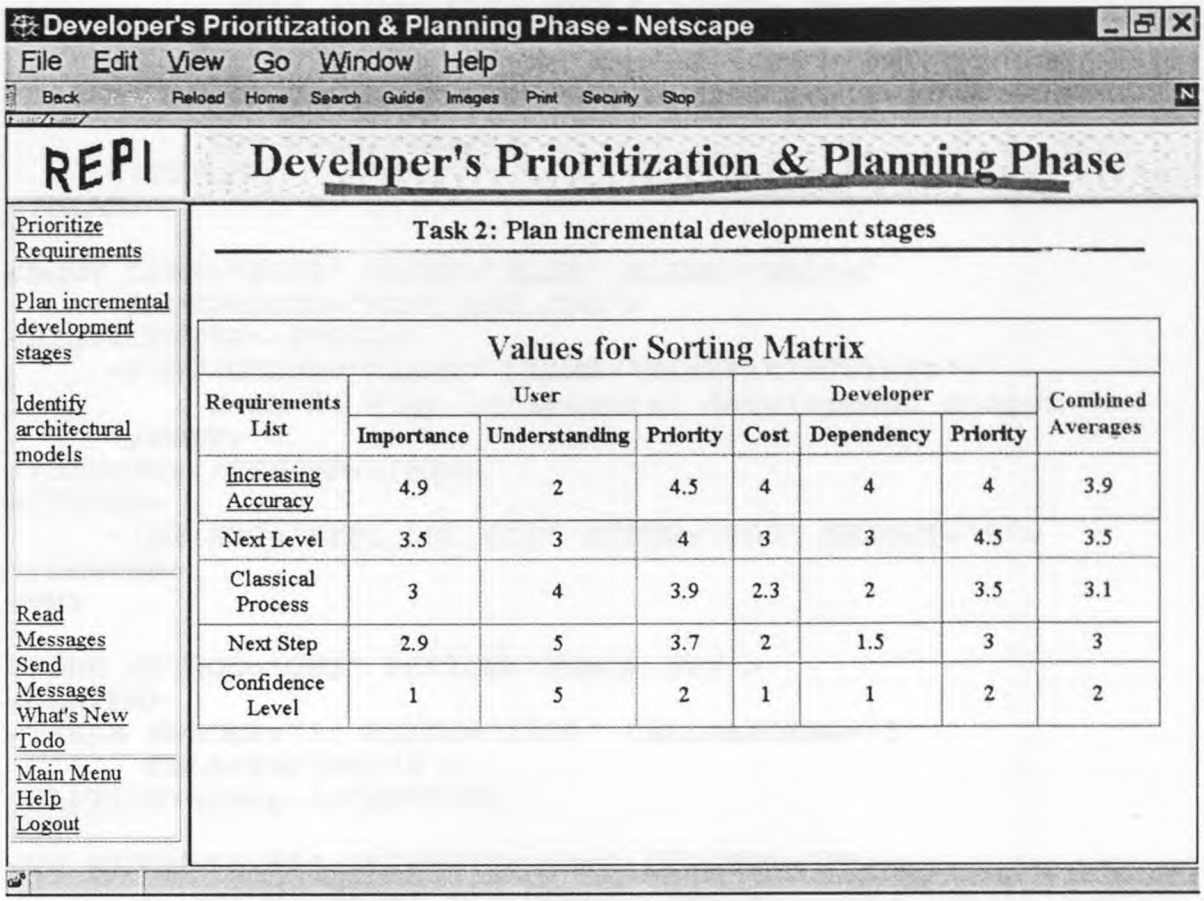


Figure 77: Task 2 of the Developer's Prioritization and Planning Phase

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">

<!-- REPI Web Site                                          -->
<!-- Demo Version - November 24, 1997                       -->
<!-- Copyright (c) 1997                                     -->
<!--                                                        -->
<!-- Author: Deepak Pandit                                  -->
<!-- Email: dpandit@hotmail.com  or
     dnp3128@megahertz.njit.edu                             -->


<HTML>

<HEAD>
     <META NAME="Author" CONTENT="Deepak Pandit">
     <META HTTP-EQUIV="Content-Type"
          CONTENT="text/html;CHARSET=iso-8859-1">
```

```
      <LINK REL=STYLESHEET TYPE="text/javascript"
            HREF="REPI.CSS" TITLE="Style Sheet">

      <TITLE>
           Developer's Prioritization & Planning Task 2: Sort
           the Requirements
      </TITLE>
</HEAD>

<BODY LINK="BLUE" VLINK="BLUE" ALINK="White"
       BACKGROUND="REPI_BK2.JPG">
<BIG><CENTER><STRONG>
      <FONT COLOR="Black" CLASS="TaskTitleFormat">
           Task 2: Plan incremental development stages
      </FONT>
</STRONG></CENTER></BIG>
<CENTER>
      <IMG SRC="REPI_LN1.GIF" WIDTH="800" HEIGHT="5">
</CENTER>
<BR>

<FORM METHOD="GET" ACTION="ERROR.HTM">
<CENTER>
<TABLE BORDER="1" WIDTH="100%" CELLPADDING="5"
        CELLSPACING="0">
<CAPTION> </CAPTION>
<TR>
<TH WIDTH="128%" ALIGN="Center" COLSPAN="8">
           <FONT SIZE="+2" COLOR="Red">
                Values for Sorting Matrix
           </FONT>
      </TH>
</TR>
<TR>
      <TH WIDTH="14%" ALIGN="Center" ROWSPAN="2">
           <FONT COLOR="Red">Requirements List</FONT>
      </TH>
      <TH WIDTH="42%" ALIGN="Center" COLSPAN="3">
           <FONT COLOR="Red">User</FONT>
      </TH>
      <TH WIDTH="44%" ALIGN="Center" COLSPAN="3">
           <FONT COLOR="Red">Developer</FONT>
      </TH>
      <TH WIDTH="15%" ALIGN="Center" ROWSPAN="2">
           <FONT COLOR="Red">Combined Averages</FONT>
      </TH>
</TR>
<TR>
      <TH WIDTH="14%" ALIGN="Center">Importance</TH>
      <TH WIDTH="14%" ALIGN="Center">Understanding</TH>
      <TH WIDTH="14%" ALIGN="Center">Priority</TH>
      <TH WIDTH="15%" ALIGN="Center">Cost</TH>
      <TH WIDTH="15%" ALIGN="Center">Dependency</TH>
      <TH WIDTH="14%" ALIGN="Center">Priority</TH>
```

```
</TR>
<TR>
     <TD WIDTH="14%" ALIGN="Center">
          <A HREF="RINFO_1.HTM">Increasing Accuracy</A>
     </TD>
     <TD WIDTH="14%" ALIGN="Center">4.9</TD>
     <TD WIDTH="14%" ALIGN="Center">2</TD>
     <TD WIDTH="14%" ALIGN="Center">4.5</TD>
     <TD WIDTH="15%" ALIGN="Center">4</TD>
     <TD WIDTH="15%" ALIGN="Center">4</TD>
     <TD WIDTH="14%" ALIGN="Center">4</TD>
     <TD WIDTH="14%" ALIGN="Center">3.9</TD>
</TR>
<TR>
     <TD WIDTH="14%" ALIGN="Center">Next Level</TD>
     <TD WIDTH="14%" ALIGN="Center">3.5</TD>
     <TD WIDTH="14%" ALIGN="Center">3</TD>
     <TD WIDTH="14%" ALIGN="Center">4</TD>
     <TD WIDTH="15%" ALIGN="Center">3</TD>
     <TD WIDTH="15%" ALIGN="Center">3</TD>
     <TD WIDTH="14%" ALIGN="Center">4.5</TD>
     <TD WIDTH="14%" ALIGN="Center">3.5</TD>
</TR>
<TR>
     <TD WIDTH="14%" ALIGN="Center">Classical Process</TD>
     <TD WIDTH="14%" ALIGN="Center">3</TD>
     <TD WIDTH="14%" ALIGN="Center">4</TD>
     <TD WIDTH="14%" ALIGN="Center">3.9</TD>
     <TD WIDTH="15%" ALIGN="Center">2.3</TD>
     <TD WIDTH="15%" ALIGN="Center">2</TD>
     <TD WIDTH="14%" ALIGN="Center">3.5</TD>
     <TD WIDTH="14%" ALIGN="Center">3.1</TD>
</TR>
<TR>
     <TD WIDTH="14%" ALIGN="Center">Next Step</TD>
     <TD WIDTH="14%" ALIGN="Center">2.9</TD>
     <TD WIDTH="14%" ALIGN="Center">5</TD>
     <TD WIDTH="14%" ALIGN="Center">3.7</TD>
     <TD WIDTH="15%" ALIGN="Center">2</TD>
     <TD WIDTH="15%" ALIGN="Center">1.5</TD>
     <TD WIDTH="14%" ALIGN="Center">3</TD>
     <TD WIDTH="14%" ALIGN="Center">3</TD>
</TR>
<TR>
     <TD WIDTH="14%" ALIGN="Center">Confidence Level</TD>
     <TD WIDTH="14%" ALIGN="Center">1</TD>
     <TD WIDTH="14%" ALIGN="Center">5</TD>
     <TD WIDTH="15%" ALIGN="Center">2</TD>
     <TD ALIGN="Center">1</TD>
     <TD ALIGN="Center">1</TD>
     <TD ALIGN="Center">2</TD>
     <TD WIDTH="14%" ALIGN="Center">2</TD>
</TR>
</TABLE>
```

```
</FORM>
</BODY>

</HTML>
```

Figure 78: HTML Source Code for Task 2 of Developer's Prioritization and Planning Phase

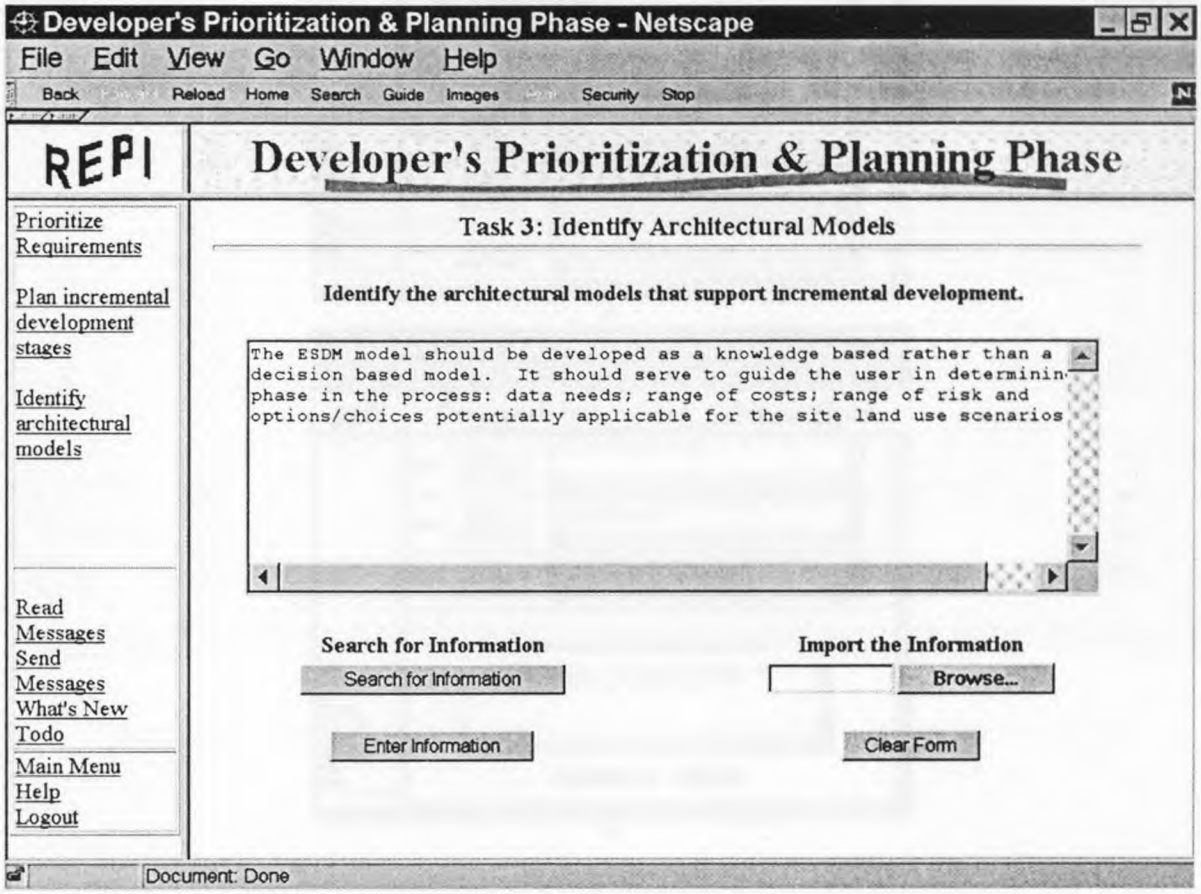


Figure 79: Task 3 of the Developer's Prioritization and Planning Phase

I.5: DEVELOPER'S INTEGRATION and VALIDATION PAGES

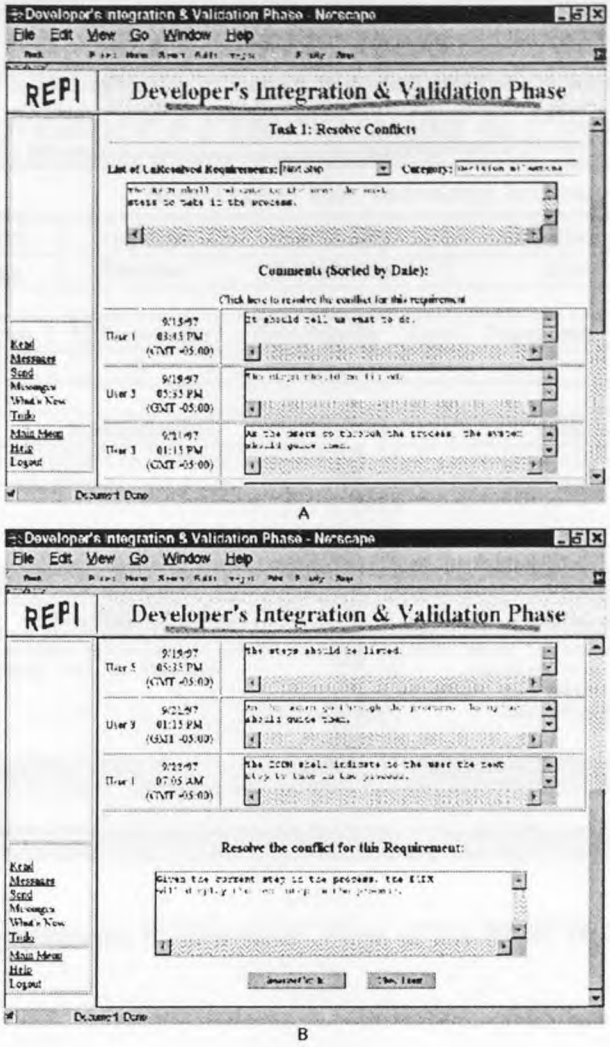


Figure 80: Task 1 of the Developer's Integration and Validation Phase

J.1: REPI WEB SITE's INFORMATION PAGES

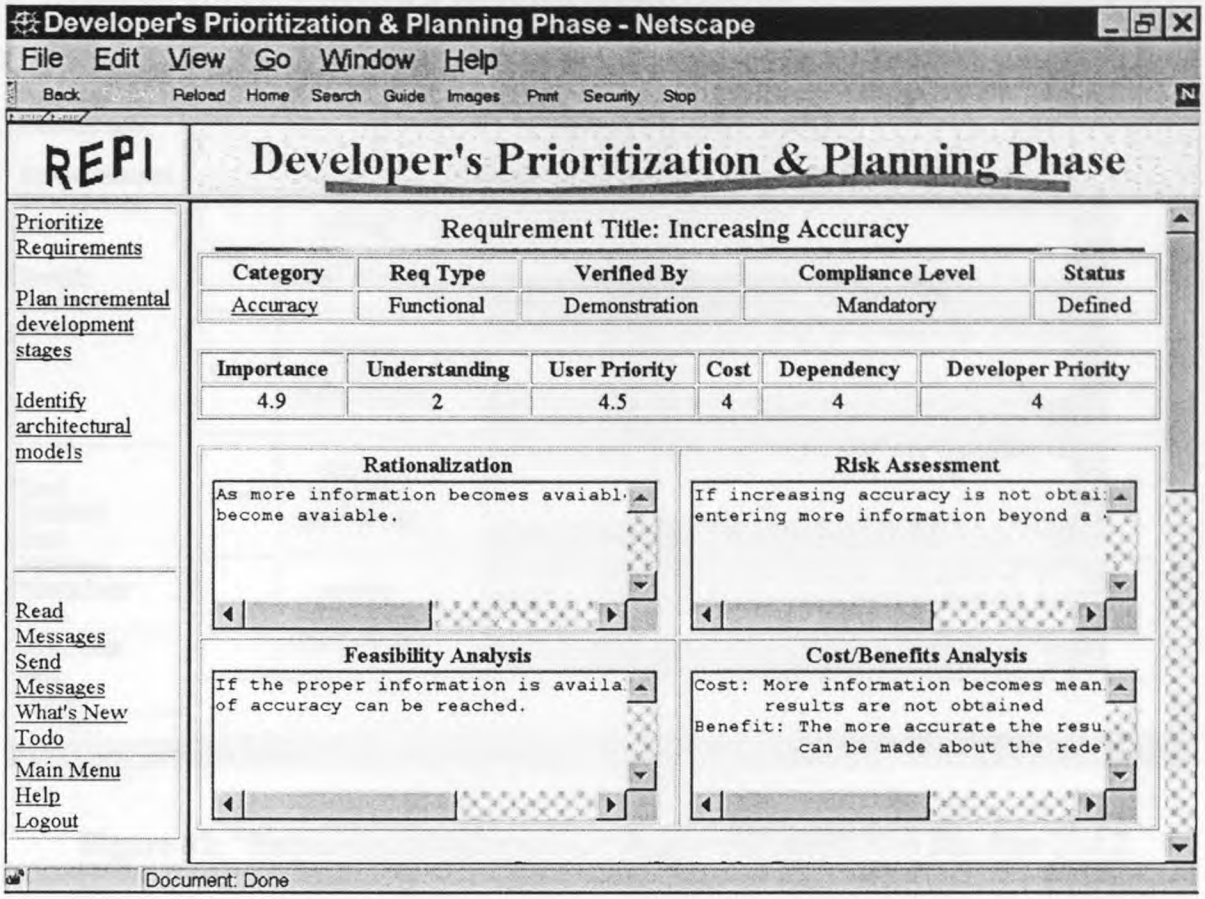


Figure 81: "Requirements Information" Page of the REPI Web Site Part 1 of 2

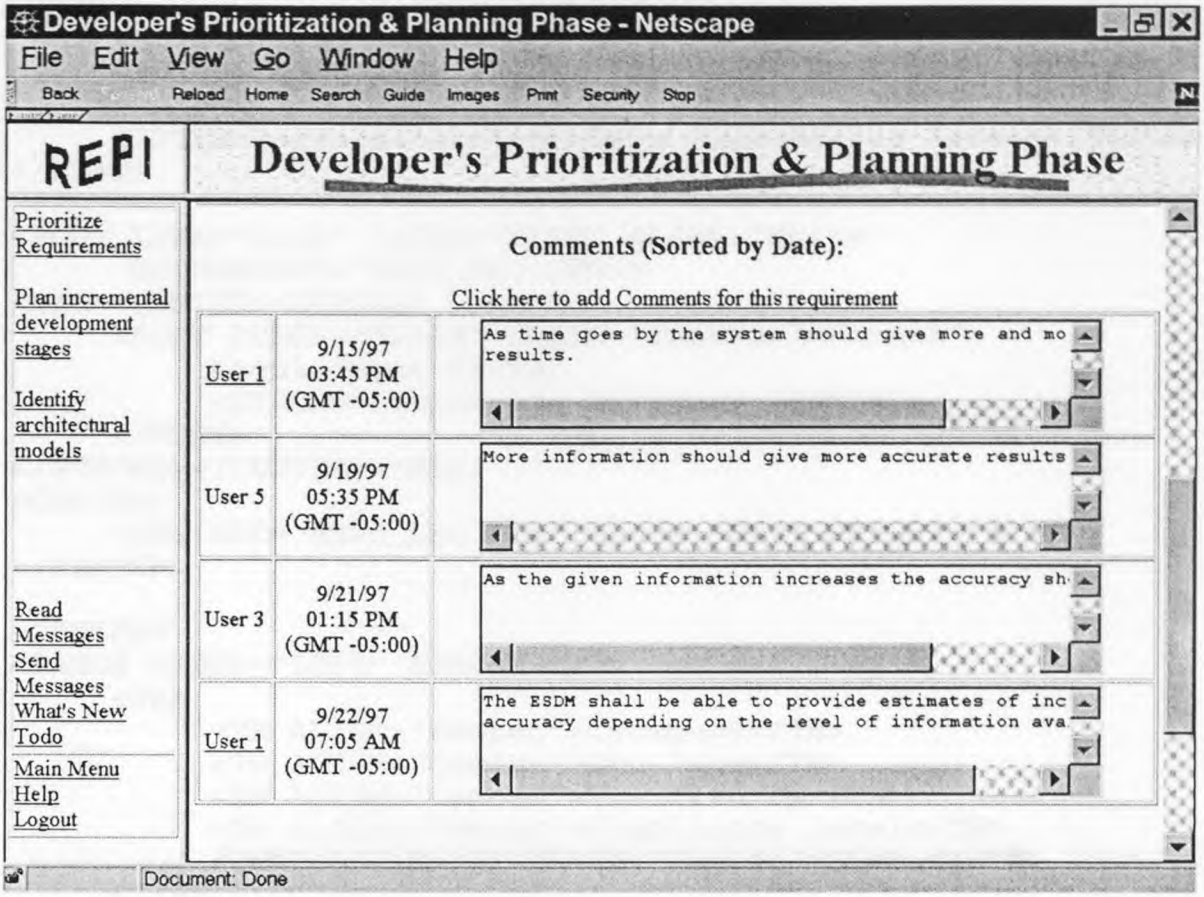


Figure 82: "Requirements Information" Page of the REPI Web Site Part 2 of 2

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">

<!-- REPI Web Site                                          -->
<!-- Demo Version - November 24, 1997                       -->
<!-- Copyright (c) 1997                                     -->
<!--                                                        -->
<!-- Author: Deepak Pandit                                  -->
<!-- Email: dpandit@hotmail.com  or
     dnp3128@megahertz.njit.edu                             -->


<HTML>

<HEAD>
     <META NAME="Author" CONTENT="Deepak Pandit">
     <META HTTP-EQUIV="Content-Type"
          CONTENT="text/html;CHARSET=iso-8859-1">

     <LINK REL=STYLESHEET TYPE="text/javascript"
          HREF="REPI.CSS" TITLE="Style Sheet">
```

```
     <SCRIPT LANGUAGE="JavaScript" SRC="REPI.JS"></SCRIPT>

     <TITLE>Increasing Accuracy's Information Screen</TITLE>
</HEAD>

<BODY LINK="BLUE" VLINK="BLUE" ALINK="White"
      BACKGROUND="REPI_BK2.JPG">
<BIG><CENTER><STRONG>
     <FONT COLOR="Black" CLASS="TaskTitleFormat">
          Requirement Title:
          <STRONG>Increasing Accuracy</STRONG>
     </FONT>
</STRONG></CENTER></BIG>
<CENTER>
     <IMG SRC="REPI_LN1.GIF" WIDTH="800" HEIGHT="5">
</CENTER>

<CENTER>
<TABLE WIDTH="100%" BORDER="1">
     <TR>
          <TH ALIGN="Center">Category</TH>
          <TH ALIGN="Center">Req Type</TH>
          <TH ALIGN="Center">Verified By</TH>
          <TH ALIGN="Center">Compliance Level</TH>
          <TH COLSPAN="2" ALIGN="Center">Status</TH>
     </TR>
     <TR>
          <TD ALIGN="Center">
               <A HREF="CINFO_1.HTM"
                  TARGET="Right">Accuracy</A>
          </TD>
          <TD ALIGN="Center">Functional</TD>
          <TD ALIGN="Center">Demonstration</TD>
          <TD ALIGN="Center">Mandatory</TD>
          <TD COLSPAN="2" ALIGN="Center">Defined</TD>
     </TR>
</TABLE>
<BR>
<TABLE WIDTH="100%" BORDER="1">
     <TR>
          <TH ALIGN="Center">Importance</TH>
          <TH ALIGN="Center">Understanding</TH>
          <TH ALIGN="Center">User Priority</TH>
          <TH ALIGN="Center">Cost</TH>
          <TH ALIGN="Center">Dependency</TH>
          <TH ALIGN="Center">Developer Priority</TH>
     </TR>
     <TR>
          <TD ALIGN="Center">4.9</TD>
          <TD ALIGN="Center">2</TD>
          <TD ALIGN="Center">4.5</TD>
          <TD ALIGN="Center">4</TD>
          <TD ALIGN="Center">4</TD>
```

```
          <TD ALIGN="Center">4</TD>
     </TR>
</TABLE>
<BR>
<FORM METHOD="GET" ACTION="ERROR.HTM">
<TABLE ALIGN="Center" WIDTH="100%" BORDER="1">
<TR>
     <TH ALIGN="Center">
          Rationalization<BR>
          <TEXTAREA NAME="Rationalization" COLS="35"
                    ROWS="5">
As more information becomes avaiable, more accurate results
should become avaiable.
          </TEXTAREA>
     </TH>
     <TH ALIGN="Center">
          Risk Assessment<BR>
          <TEXTAREA NAME="Risks" COLS="35" ROWS="5">
If increasing accuracy is not obtained then users will stop
entering more information beyond a certain point.
          </TEXTAREA>
     </TH>
</TR>
<TR>
     <TH ALIGN="Center">
          Feasibility Analysis<BR>
          <TEXTAREA NAME="Feasibility" COLS="35" ROWS="5">
If the proper information is available then the next level
of accuracy can be reached.
          </TEXTAREA>
     </TH>
     <TH ALIGN="Center">
          Cost/Benefits Analysis<BR>
          <TEXTAREA NAME="Cost_Benefit" COLS="35" ROWS="5">
Cost: More information becomes meaning less if more accurate
      results are not obtained
Benefit: The more accurate the results the better decision
         can be made about the redevelopment options.
          </TEXTAREA>
     </TH>
</TR>
</TABLE>
<BR>
<H3>Comments (Sorted by Date):</H3>
<A HREF="#Additional Comments">
     Click here to add Comments for this requirement
</A>

<TABLE WIDTH="100%" BORDER="1">
<TR>
     <TD ALIGN="Center">
          <A HREF="UINFO_1.HTM#R1_U1_1">User 1</A>
     </TD>
     <TD ALIGN="Center">
```

```
          9/15/97<BR>03:45 PM<BR>(GMT -05:00)
     </TD>
     <TD ALIGN="Center">
          <TEXTAREA NAME="User_1_Comment_1" ROWS="3"
                    COLS="50">
As time goes by the system should give more and more
accurate results.
          </TEXTAREA>
     </TD>
</TR>
<TR>
     <TD ALIGN="Center">User 5</TD>
     <TD ALIGN="Center">
          9/19/97<BR>05:35 PM<BR>(GMT -05:00)
     </TD>
     <TD ALIGN="Center">
          <TEXTAREA NAME="User_5_Comment_1" ROWS="3"
                    COLS="50">
More information should give more accurate results
          </TEXTAREA>
     </TD>
</TR>
<TR>
     <TD ALIGN="Center">User 3</TD>
     <TD ALIGN="Center">
          9/21/97<BR>01:15 PM<BR>(GMT -05:00)
     </TD>
     <TD ALIGN="Center">
          <TEXTAREA NAME="User_3_Comment_1" ROWS="3"
                    COLS="50">
As the given information increases the accuracy should
increase
          </TEXTAREA>
     </TD>
</TR>
<TR>
     <TD ALIGN="Center">
          <A HREF="UINFO_1.HTM#R1_U1_2">User 1</A>
     </TD>
     <TD ALIGN="Center">
          9/22/97<BR>07:05 AM<BR>(GMT -05:00)
     </TD>
     <TD ALIGN="Center">
          <TEXTAREA NAME="User_1_Comment_1" ROWS="3"
                    COLS="50">
The ESDM shall be able to provide estimates of increasing
accuracy depending on the level of information available.
          </TEXTAREA>
     </TD>
</TR>
</TABLE>
<BR><BR>
<A NAME="Additional Comments"></A>
<H3>Additional Comments about this Requirement:</H3>
```

```
<TEXTAREA NAME="Comments" ROWS="3" COLS="60"></TEXTAREA>
<BR><BR>
<INPUT TYPE="Button" NAME="RInfo_1_Enter"
       VALUE="Enter Comments" onClick="open_error()">
     
<INPUT TYPE="Reset" VALUE="Clear Form">
</FORM>
</CENTER>
</BODY>

</HTML>
```

Figure 83: HTML Source Code for the "Requirements Information" Page

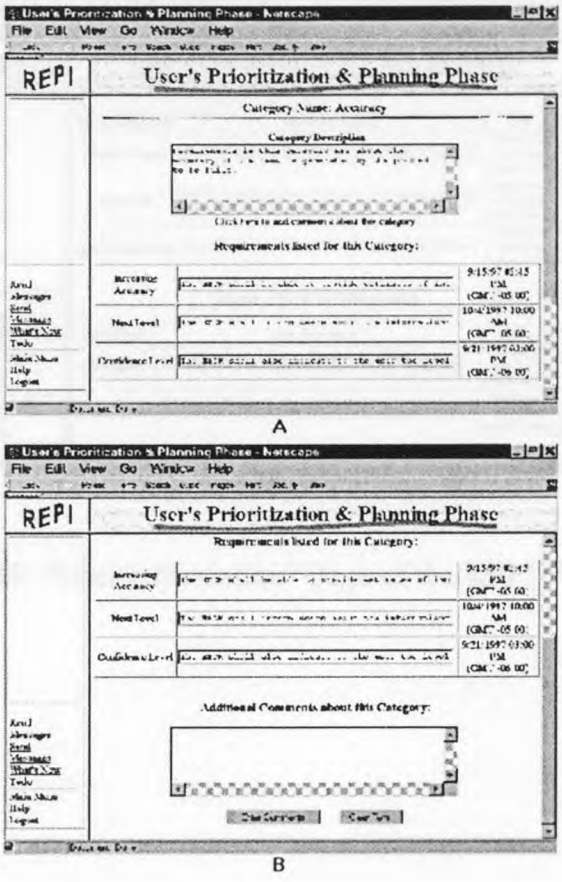


Figure 84: "Category Information" Page of the REPI Web Site

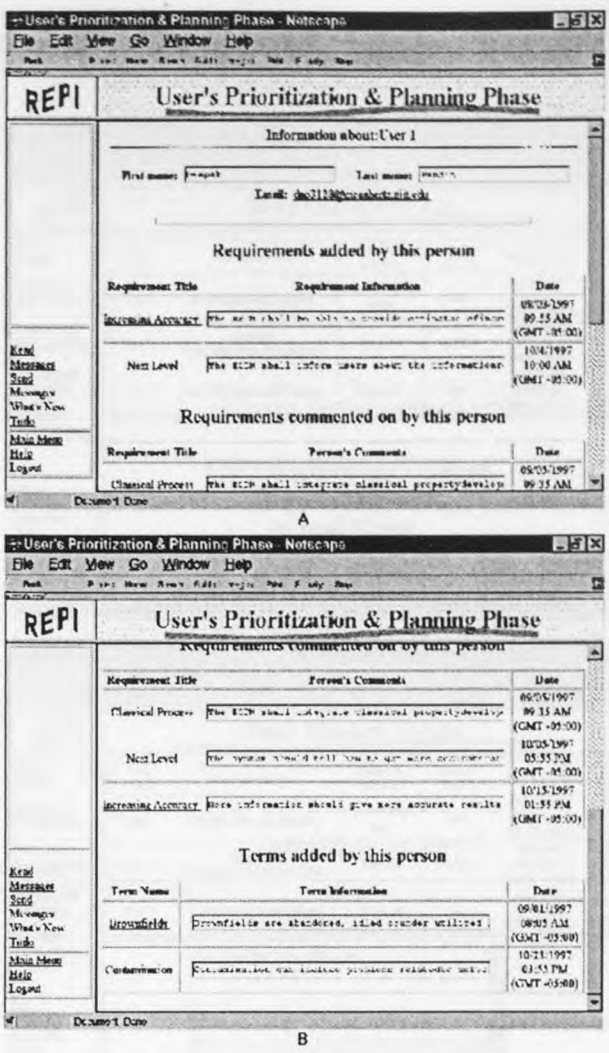


Figure 85: "User Information" Page of the REPI Web Site

K.1: READ AND SEND MESSAGES PAGES

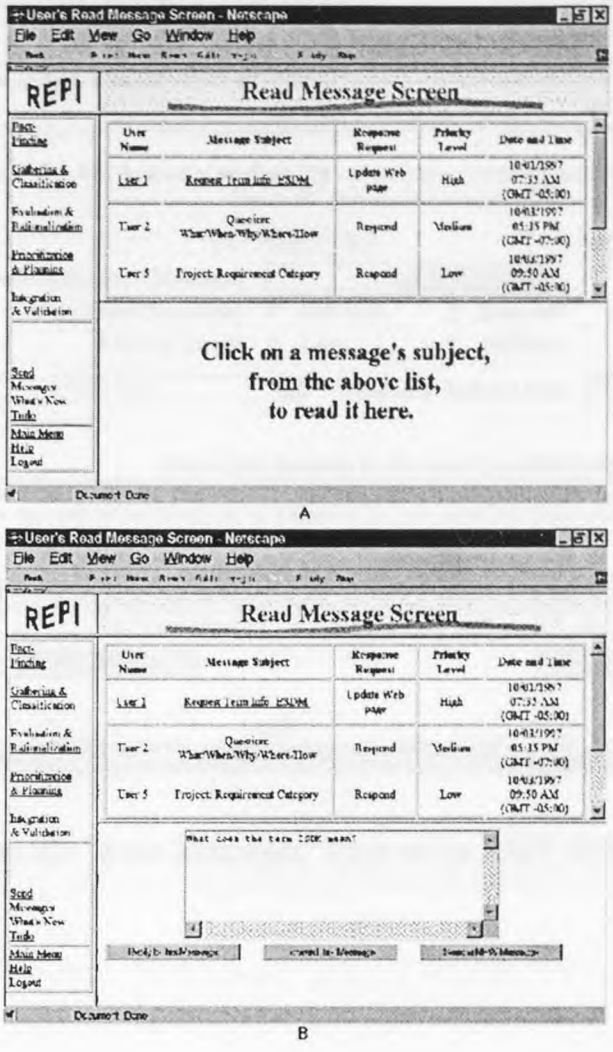


Figure 86: "Read Messages" Page of the REPI Web Site

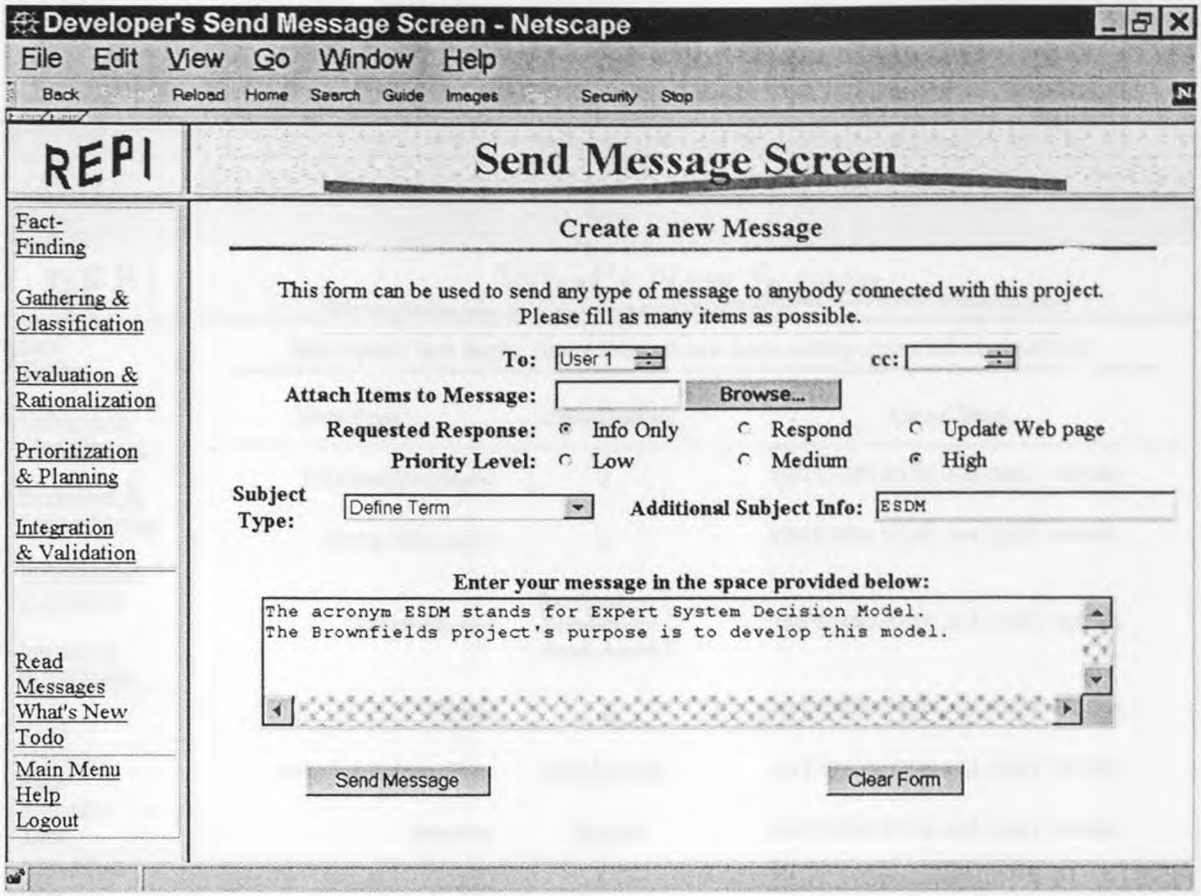


Figure 87: "Send Messages" Page of the REPI Web Site

K.2: WHAT'S NEW PAGE

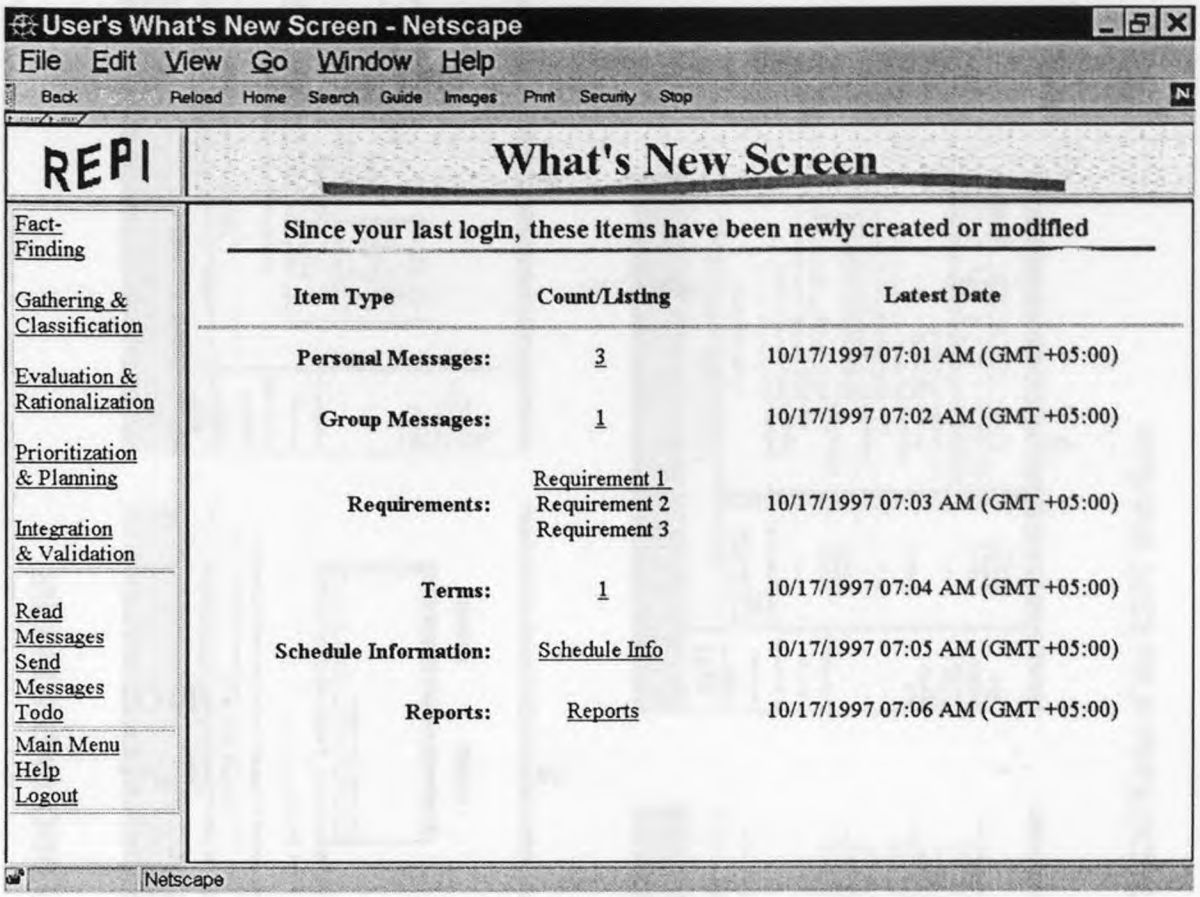


Figure 88: "What's New" Page of the REPI Web Site

K.3: TODO TASKS PAGES

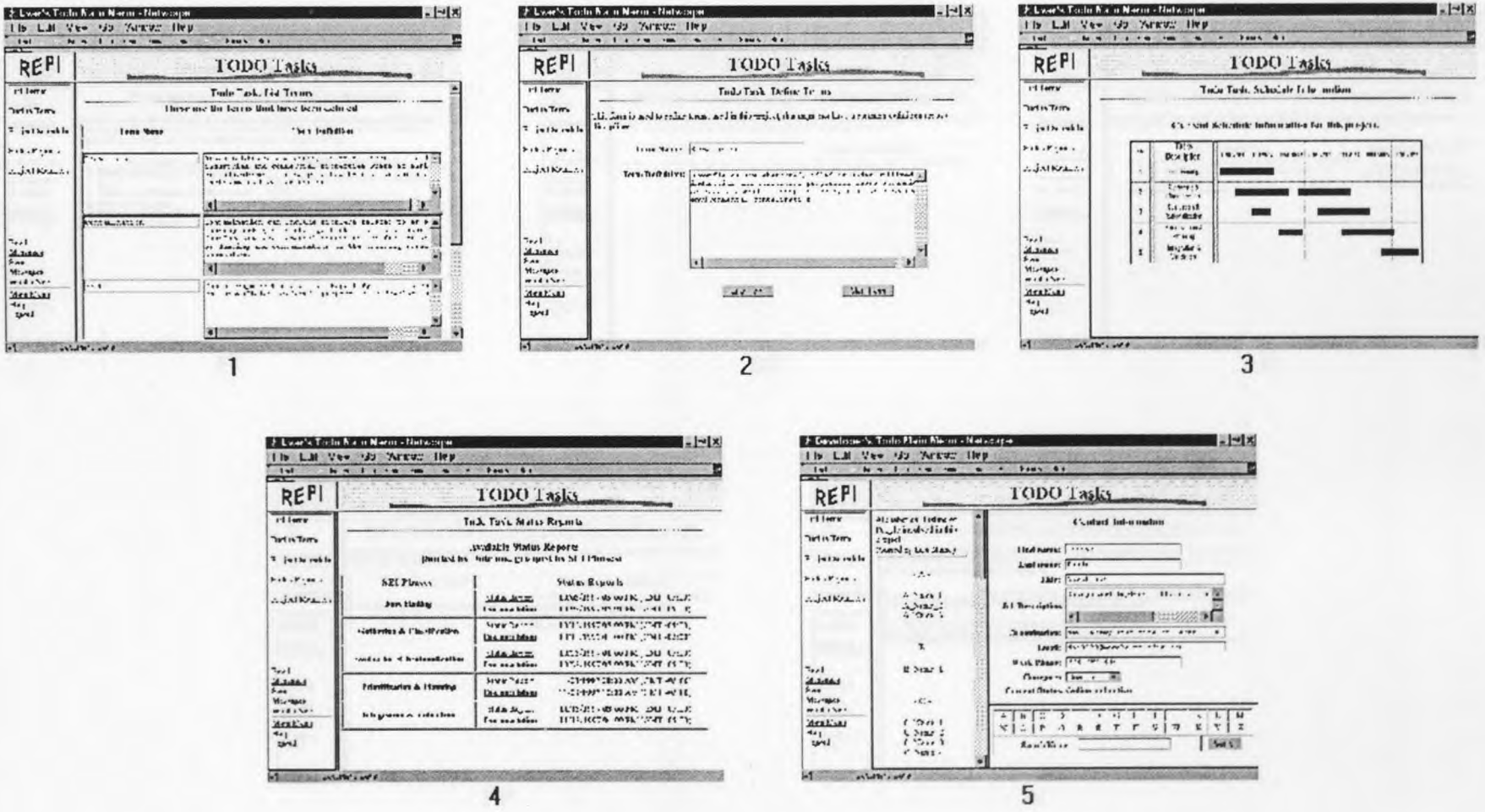


Figure 89: Five TODO Tasks of the REPI Web Ssite

K.4: HELP PAGES

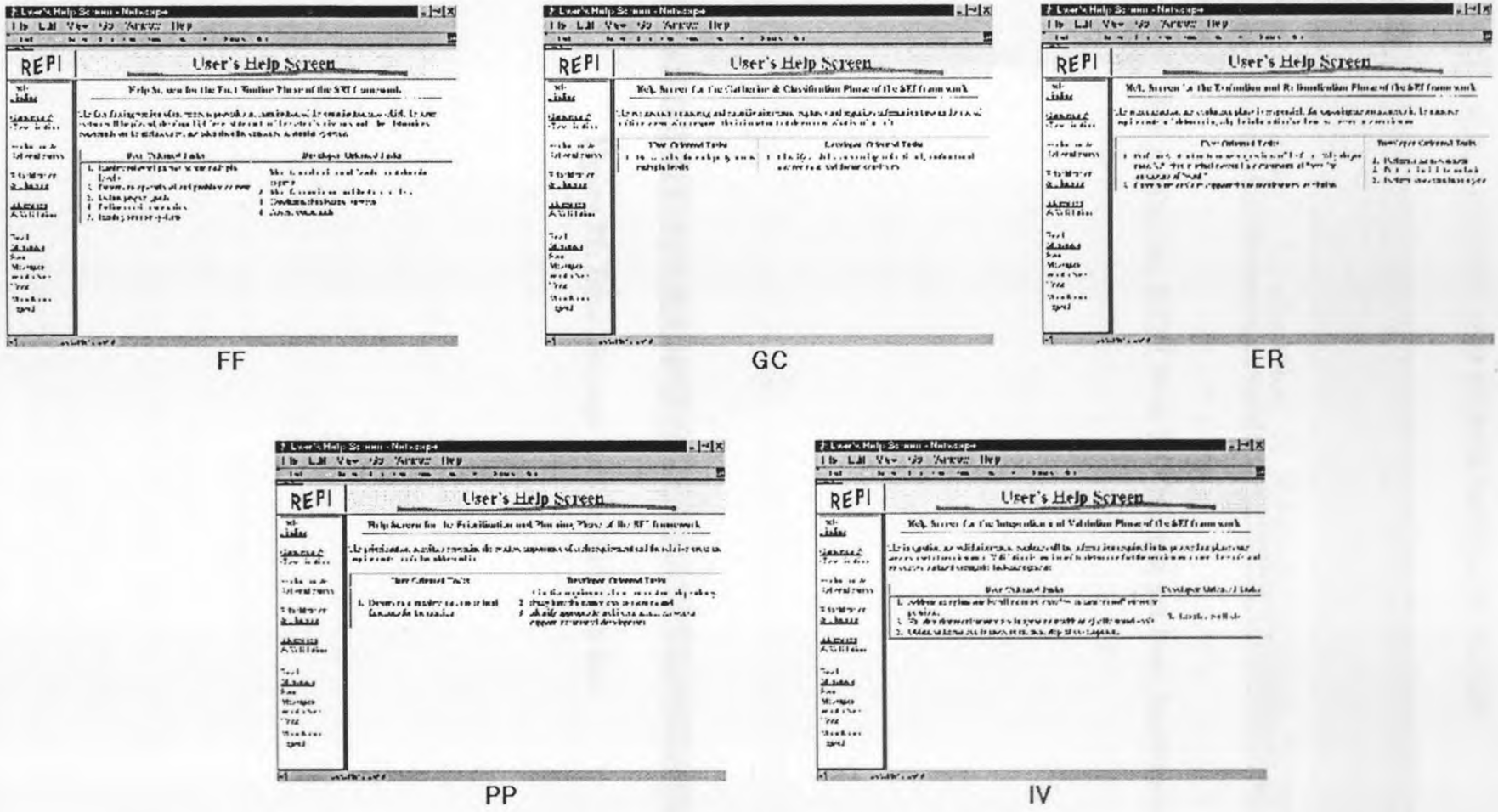


Figure 90: Help Pages for the REPI Web Site

K.5: LOGOUT AND ERROR MESSAGE PAGES

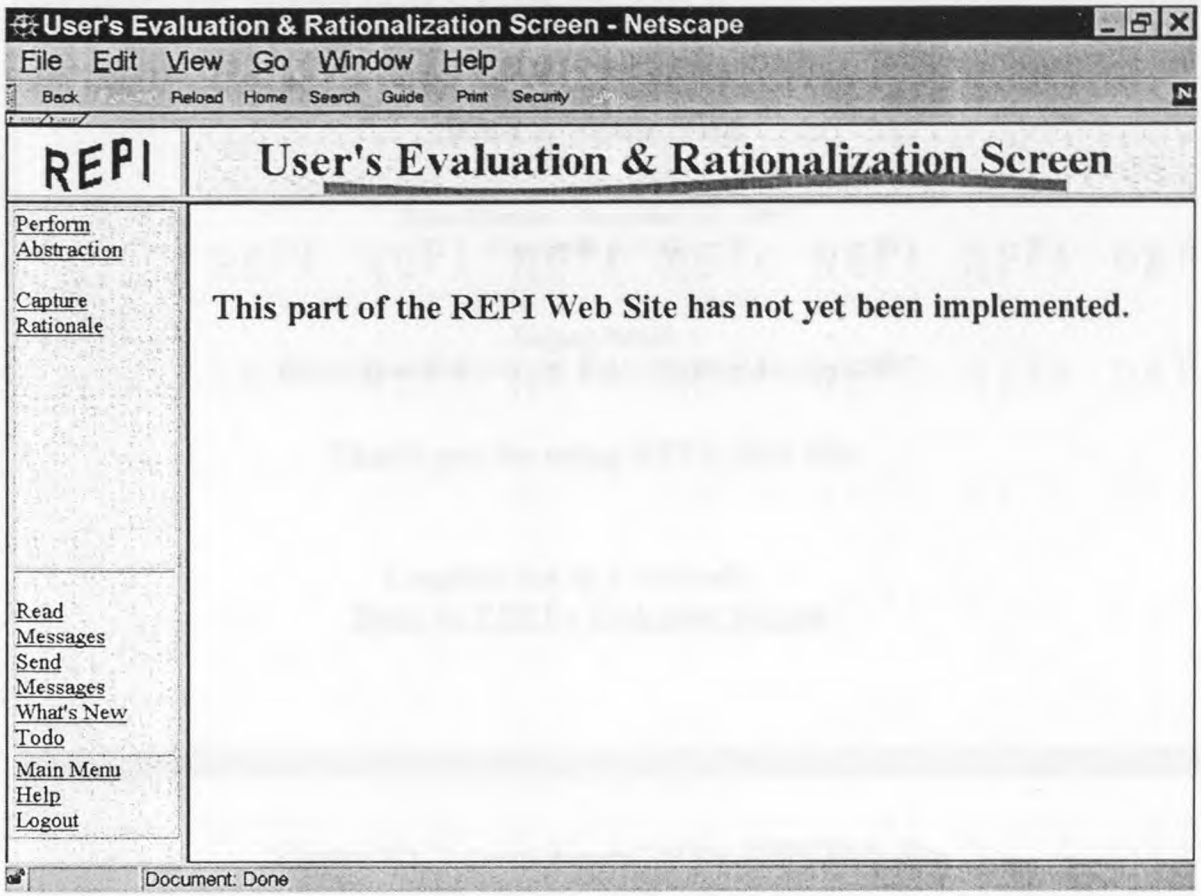


Figure 91: Error Message for the REPI Web Site

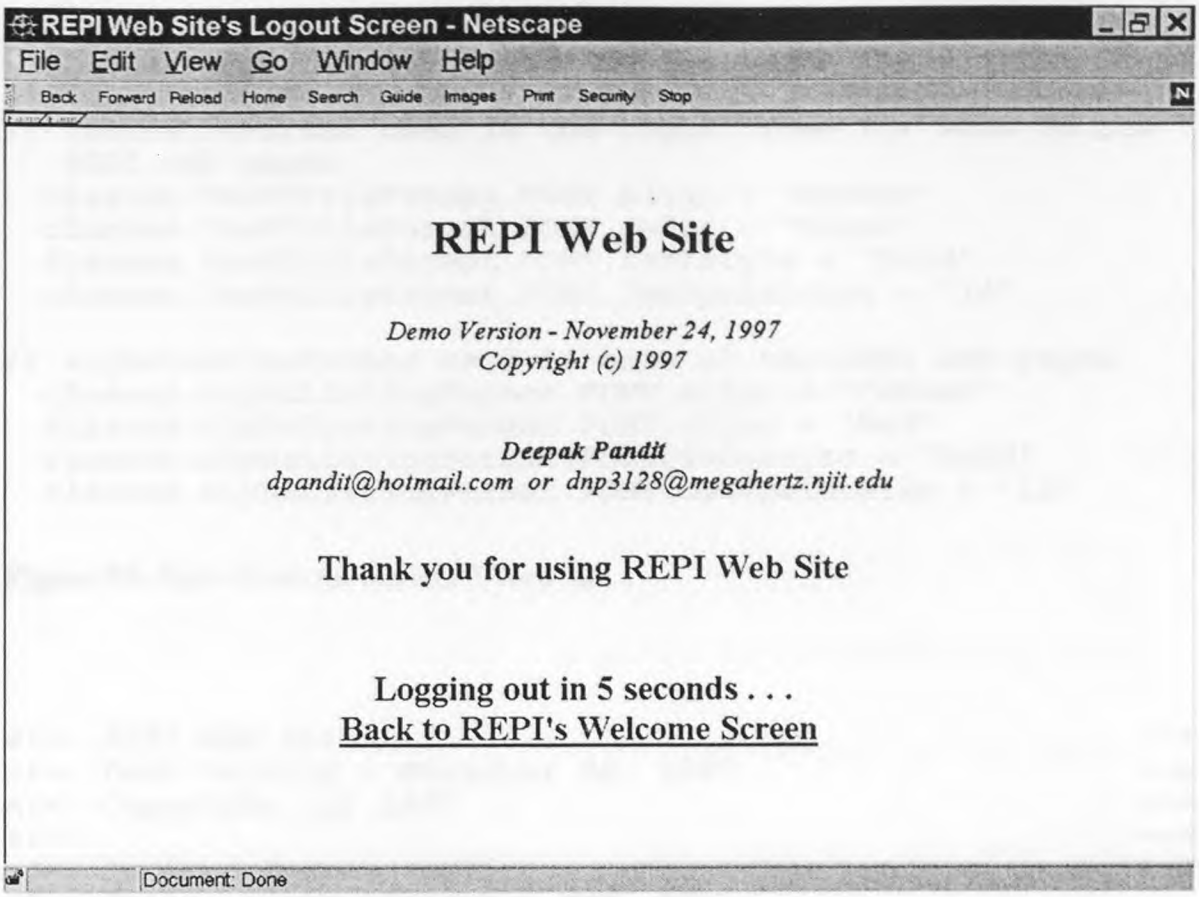


Figure 92: "Logout Screen" of the REPI Web Site

L.1: REPI WEB SITE's STYLE SHEET AND JAVASCRIPT CODE

```
<!-- REPI Web Site                                          -->
<!-- Demo Version - November 24, 1997                       -->
<!-- Copyright (c) 1997                                     -->
<!--                                                        -->
<!-- Author: Deepak Pandit                                  -->
<!-- Email: dpandit@hotmail.com  or
     dnp3128@megahertz.njit.edu                             -->

<!-- Style Sheet for the Thesis REPI Web Site -->
<!-- Created by Deepak Pandit                 -->

// ScreenTitleFormat used in the title frame of all REPI web
   pages
  classes.ScreenTitleFormat.FONT.align = "Center"
  classes.ScreenTitleFormat.FONT.color = "Red"
  classes.ScreenTitleFormat.FONT.font = "TimesRoman"
```

```
  classes.ScreenTitleFormat.FONT.fontstyle = "Bold"
  classes.ScreenTitleFormat.FONT.fontpointsize = "12"

// TaskTitleFormat used in the right frame for some of the
   REPI web pages
  classes.TaskTitleFormat.FONT.align = "Center"
  classes.TaskTitleFormat.FONT.color = "Black"
  classes.TaskTitleFormat.FONT.fontstyle = "Bold"
  classes.TaskTitleFormat.FONT.fontpointsize = "14"

// AlphaListingFormat used in some of the REPI web pages
  classes.AlphaListingFormat.FONT.align = "Center"
  classes.AlphaListingFormat.FONT.color = "Red"
  classes.AlphaListingFormat.FONT.fontstyle = "Bold"
  classes.AlphaListingFormat.FONT.fontpointsize = "12"
```

Figure 93: Style Sheet for the REPI Web Site

```
<!-- REPI Web Site                                          -->
<!-- Demo Version - November 24, 1997                       -->
<!-- Copyright (c) 1997                                     -->
<!--                                                        -->
<!-- Author: Deepak Pandit                                  -->
<!-- Email: dpandit@hotmail.com  or
     dnp3128@megahertz.njit.edu                             -->

<!-- ****************************************************** -->

//   JavaScript functions used in Thesis's REPI Web Site
//   Created by Deepak Pandit

<!-- ****************************************************** -->

<!-- ====================================================== -->
//   Functions used in menu type web pages of the REPI's Web
     Site
<!-- ====================================================== -->

function display_status (status_code)
// Displays a status message based on the status_code
   parameter
// Codes used in the HTML pages when calling this function
   are
// U_MAIN, U_READ, U_SEND, U_NEW, U_TODO, and U_HELP
// U_FF, U_GC, U_ER, U_PP, and U_IV
// U_FF_1 to U_FF_5, U_GC_1 and U_GC_2, U_ER_1 and U_ER_2,
   U_IV_1 to U_IV_3
// D_MAIN, D_READ, D_SEND, D_NEW, D_TODO, and D_HELP
// D_FF, D_GC, D_ER, D_PP, and D_IV
```

```
// D_FF_1 to D_FF_4, D_GC_1 to D_GC_3, D_ER_1 to D_ER_3,
   D_PP_1 to D_PP_3
// TODO_1 to TODO_5, HELP_F, HELP_G, HELP_E, HELP_P, and
   HELP_I

 {

  // User web pages

  if (status_code == 'U_FF')
    window.status = "Examine the organization, into which,
                     the target system will be placed";
  if (status_code == 'U_FF_1')
    window.status = "Identify potential stakeholders of this
                     project";
  if (status_code == 'U_FF_2')
    window.status = "Describe the problem that is to be
                     solved by this project";
  if (status_code == 'U_FF_3')
    window.status = "List the goals to be reached by this
                     project";
  if (status_code == 'U_FF_4')
    window.status = "List the general scenarios for this
                     project";
  if (status_code == 'U_FF_5')
    window.status = "Identify other systems that are similar
                     to the system to be built";

  if (status_code == 'U_GC')
    window.status = "Capture and organize the information
                     that determines what is to be built";
  if (status_code == 'U_GC_1')
    window.status = "List of available requirements";
  if (status_code == 'U_GC_2')
    window.status = "Add a new requirement";

  if (status_code == 'U_ER')
    window.status = "Expose inconsistencies in the gathered
                     requirements and determining why the
                     information has been expressed as a
                     requirement";
  if (status_code == 'U_ER_1')
    window.status = "Answer why you need these
                     requirements";
  if (status_code == 'U_ER_2')
    window.status = "Describe the reasons for these
                     requirements";

  if (status_code == 'U_PP')
    window.status = "Determine the relative importance of
                     each requirement";

  if (status_code == 'U_IV')
```

```
  window.status = "Identify missing requirements and
                   verify they meet the goals";
if (status_code == 'U_IV_1')
  window.status = "Define 'To Be Determined'
                   Requirements";
if (status_code == 'U_IV_2')
  window.status = "Verify that requirements are in
                   agreement with the goals";
if (status_code == 'U_IV_3')
  window.status = "Authorize the developers to move to the
                   next step";

if (status_code == 'U_READ')
  window.status = "User's Read Messages Screen";
if (status_code == 'U_SEND')
  window.status = "User's Send Messages Screen";
if (status_code == 'U_NEW')
  window.status = "User's What's New Screen";
if (status_code == 'U_TODO')
  window.status = "User's TODO Screen";
if (status_code == 'U_MAIN')
  window.status = "User's Main Menu Screen";
if (status_code == 'U_HELP')
  window.status = "User's Help Menu Screen";

// Developer web pages

if (status_code == 'D_FF')
  window.status = "Examine the technological and
                   developmental issues of the target
                   system";
if (status_code == 'D_FF_1')
  window.status = "Identify the domain experts for this
                   project";
if (status_code == 'D_FF_2')
  window.status = "Identify the domain models for this
                   project";
if (status_code == 'D_FF_3')
  window.status = "Enter the technological survey
                   information";
if (status_code == 'D_FF_4')
  window.status = "Provide reasons for the given
                   constraints";

if (status_code == 'D_GC')
  window.status = "Classify the available information and
                   add new information";
if (status_code == 'D_GC_1')
  window.status = "Categorize the requirements";
if (status_code == 'D_GC_2')
  window.status = "List of available requirements";
if (status_code == 'D_GC_3')
  window.status = "Add a new requirement";
```

```
if (status_code == 'D_ER')
  window.status = "Evaluate the various risks associated
                   with each requirement";
if (status_code == 'D_ER_1')
  window.status = "Assess the risk for the given
                   requirements";
if (status_code == 'D_ER_2')
  window.status = "Analyze the feasibility for the given
                   requirements";
if (status_code == 'D_ER_3')
  window.status = "Analyze the Costs and Benefits for the
                   given requirements";

if (status_code == 'D_PP')
  window.status = "Determine the relative order the
                   requirements should be addressed in";
if (status_code == 'D_PP_1')
  window.status = "Prioritize requirements by Cost and
                   Dependency";
if (status_code == 'D_PP_2')
  window.status = "Sort requirements for developing harder
                   requirements first";
if (status_code == 'D_PP_3')
  window.status = "Identify the models for incremental
                   development";

if (status_code == 'D_IV')
  window.status = "Identify and resolve the conflicts
                   between existing requirements";

if (status_code == 'D_READ')
  window.status = "Developer's Read Messages Screen";
if (status_code == 'D_SEND')
  window.status = "Developer's Send Messages Screen";
if (status_code == 'D_NEW')
  window.status = "Developer's What's New Screen";
if (status_code == 'D_TODO')
  window.status = "Developer's TODO Screen";
if (status_code == 'D_MAIN')
  window.status = "Developer's Main Menu Screen";
if (status_code == 'D_HELP')
  window.status = "Developer's Help Menu Screen";

// Common web pages

if (status_code == 'TODO_1')
  window.status = "List all the terms defined for this
                   project";
if (status_code == 'TODO_2')
  window.status = "Define new terms for this project";
if (status_code == 'TODO_3')
  window.status = "Current schedule for this project";
if (status_code == 'TODO_4')
```

```
    window.status = "Status reports available for this
                    project";
  if (status_code == 'TODO_5')
    window.status = "List of all the current members for
                    this project";

  if (status_code == 'HELP_F')
    window.status = "Help for the Fact Finding phase";
  if (status_code == 'HELP_G')
    window.status = "Help for the Gathering and
                    Classification phase";
  if (status_code == 'HELP_E')
    window.status = "Help for the Evaluation and
                    Rationalization phase";
  if (status_code == 'HELP_P')
    window.status = "Help for the Prioritization and
                    Planning phase";
  if (status_code == 'HELP_I')
    window.status = "Help for the Integration and Validation
                    phase";

  if (status_code == 'LOGIN')       // Login Screen
    window.status = "Login to REPI's Web Site";
  if (status_code == 'LOGOUT')      // Logout screen
    window.status = "Logout Screen";

  return true;
 } // display_status

<!-- ..................................................... -->

function default_status ()
// Restore the status line message to the default message
 {
  window.status = '';
  return true;
 } // default_status

<!-- ================================================== -->
//    Function used in MINFO_1.HTM web page of the REPI's Web
      Site
<!-- ================================================== -->

function open_send_msg()
// Opens the Developer's Send Message page
 {
  window.top.location.href = "D_SEND.HTM";
  return true;
 } // open_send_msg

<!-- ================================================== -->
//    Function used in form type web pages of the REPI's Web
      Site
<!-- ================================================== -->
```

```
function open_error()
// Opens the REPI Web Site's Error message page.
 {
  window.top.Right.location.href = "ERROR.HTM";
  return true;
 } // open_error

<!-- ************************************************** -->
```

Figure 94: JavaScript Source Code for the REPI Web Site

REFERENCES

[AL-RAWAS 96] Al-Rawas, Amer and Easterbrook, Steve. "Communication Problems in Requirements Engineering: A Field Study." *Proceedings of the First Westminster Conference on Professional Awareness in Software Engineering*. 1996.

[BLUNDON 96] Blundon, William. "The Truth About Java." *Internet World*. Vol. 7, No. 12. December 1996.

[BRACKETT 90] Brackett, John W. *Software Requirements*. SEI Curriculum Module SEI-CM-19-1.2. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. January 1990.

[BUTTERWORTH] Butterworth, Paul. "Web Access to the Core Business Infrastructure." Report MCS-0260-1. Forté Software Inc. September 1996.

[CGI] *Common Gateway Interface: Introduction*. http://hoohoo.ncsa.uiuc.edu/cgi/intro.html.

[CHRISTEL 92] Christel, Michael G. And Kang, Kyo C. *Issues in Requirement Elicitation*. Technical Report CMU/SEI-02-TR-12 or ESC-TR-92-012. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. September 1992.

[CLUTS 97] Cluts, Nancy Winnick. *The Dynamic HTML Object Model*. http://www.microsoft.com/intdev/ie4/omdoc-f.htm. Microsoft Corporation. April 1997.

[CONNECTED 97] "Mail Format." *Connected: An Internet Encyclopedia*. Editor Baccala, Brent. http://www.freesoft.org/Connected/Topics/05-Functions/MailFormat/index.html. 1997.

[DAVIS 93] Davis, Alan M. *Software Requirements: Objects, Functions and States*. P T R Prentice Hall, Englewood Cliffs, NJ. 1993

[ECKEL 97] Eckel, Bruce. *Thinking in Java*. http://www.EckelObjects.com. MindView Inc. 1997.

[GIRGENSOHN 96] Girgensohn, Andreas. "Experiences in Developing Collaborative Applications Using the World Wide Web 'Shell'." *Hypertext 96: The Seventh ACM Conference on Hypertext*. 1996.

[GRAY 95a] Gray, Terry. "Message Access Paradigms and Protocols." http://www.imap.org/imap.vs.pop.html. University of Washington, Seattle, WA. 1995.

[GRAY 95b] Gray, Terry. "Comparing Two Approaches to Remote Mailbox Access: IMAP vs. POP." http://www.imap.vs.pop.brief.html. University of Washington, Seattle, WA. 1995.

[GOSLING 95] Gosling, James and McGilton, Henry. "The Java Language Environment" A White Paper. Sun Microsystems, Inc. 1995.

[HARWELL 93] Harwell, Richard, et al. "What is a Requirement?" Published in the *Proceedings of the Third International Symposium of the INCOSE.* http://internet-plaza.net/incose/workgrps/rwg/what_is.html. 1993.

[HERRMANN 96] Herrmann, Eric. *Teach Yourself CGI Programming with Perl in a Week.* Sams.net, Indianapolis, Indiana. 1996.

[HONEYCUTT 97] Honeycutt, Jerry, Brown, Mark R., et al. *HTML 3.2 Starter Kit.* Que Corporation, Indianapolis, Indinana. 1997.

[HONG 97] Hong, Shuguang and Moriai, Mineo. "Evaluation Criteria for the Design of Commercial Web Sites." http://hsb.baylor.edu/ramsower/ais.ac.97/papers/hong.htm. 1997.

[IMAP4rev1] Crispin, M. "Internet Message Access Protocol - Version 4rev1." Request for Comment (RFC) 2060. http://www.internic.net/rfc/rfc2060.txt. December 1996.

[JAMSA 96] Jamsa, Kris. *Java Now!* Jamsa Press. 1996.

[KAR 96] Kar, Pradip and Bailey, Michelle. "Characteristics of Good Requirements." Paper given at the 6th INCOSE Symposium. http://gate1.tlmworks.com/cai/incose.html. 1996.

[KEIL 95] Keil, Mark and Carmel, Erran. "Customer-Developer Links in Software Development." *Communications of the ACM.* Volume 38, Number 5. Pages 38-44. May 1995.

[KRAMER 96] Kramer, Douglas. *The Java Platform: A White Paper.* Sun Microsystems, Inc. 1996.

[LEINER 97] Leiner, Barry M., Cerf, Vinton G., et al. *A Brief History of the Internet.* http://info-isoc.org/internet-history/. Feburary 1997.

[LINDEN 96] Linden, Peter van der. *just Java*. Sun Microsystems Inc. 1996.

[MICROSOFT 97a] *Dynamic HTML: The Next Generation of User Interface Design using HTML*. White Paper. http://premium.microsoft.com/msdn/library/bkgrnd/dynhtml.htm. Microsoft Corporation. February 1997.

[MICROSOFT 97b] *Technology Comparison.* http://www.microsoft.com/xxxxxxx.html. Microsoft Corporation. May 1997.

[MILLER 93] Miller, M. Greg and Tanik, Murat M. *Multimedia Applications in Software Engineering*. Technical Report 93-CSE-50. Southern Methodist University, Dallas, Texas. November 1993.

[NCSA 97] *A Beginner's Guide to HTML.* http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimerAll.html. The National Center for Supercomputing Applications. University of Illinois, Urbana-Champaign, Illinois. 1997.

[NETSCAPE 96] *The Netscape ONE Development: Environment Vision and Product*. White Paper. http://home.netscape.com/comprod/one/white_paper.html. Netscape Communications Corporation. 1996.

[NETSCAPE 97] *Persistent Client State: HTTP Cookies.* http://search.netscape.com/newsref/std/cookie_spec.html. Netscape Communications Corporation. 1997.

[NIELSEN 97] Nielsen, Henrik Frystyk and Gettys, Jim. *HTTP - Hypertext Transport Protocol*. HTTP Position Statement. W3C Architecture Domain. http://www.w3.org/Protocols/Activity.html. World Wide Web Consortium. 1997.

[OCKER 95] Ocker, Rosalie, Hiltz, Starr Roxanne, et al. "The effects of distributed group support and process structuring on software requirements." *Journal of Management Information Systems*. Winter 95/96, Volume 12, Issue 3. 1995.

[ORACLE 96] *Oracle Intranet Strategy*. An Oracle White Paper. http://www.oracle.com/promotions/intranet/html/intranet_wp.html. Oracle Corporation, Redwood Shores, CA. July 1996.

[PLAYLE 96] Playle, Greg and Schroeder, Charles. "Software Requirements Elicitation: Problems, Tools, and Techniques." http://www.stsc.hill.af.mil/crosstalk/1996/dec/xt96d12e.html. 1996.

[POHL 93] Pohl, Klaus. *The Three Dimensions of Requirement Engineering.* Technical Report NATURE-92-11. 5th International Conference on Advanced Information Systems Engineering, Paris, France. June 1993.

[RAGHAVAN 94] Raghavan, Sridhar, Zelesnik, Gregory, and Ford, Gary. *Lecture Notes on Requirements Elicitation.* Report CMU/SEI-94-EM-10. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. 1994.

[REIN 97] Rein, Lisa. "Making sense out of the latest HTML: HTML 4.0 and beyond!" http://www.netscapeworld.com/common/nw.tags.html. *NetscapeWorld.* 1997.

[RFC821] Postel, Jonathan B. "Simple Mail Transfer Protocol." Request for Comment (RFC) 821. http://ds.internic.net/rfc/rfc821.txt. 1982.

[RFC822] Crocker, David H. "Standard for the format of ARPA Internet text messages." Request for Comment (RFC) 822. http://ds.internic.net/rfc/rfc822.txt. 1982.

[RFC1869] Klensin, J., Freed, N., Rose, M., Stefferud, E., and Crocker, D. "SMTP Service Extensions." Request for Comment (RFC) 1869. http://www.internic.net/rfc/rfc1869.txt. 1995.

[RFC1939] Myers, J. and Rose, M. "Post Office Protocol - Version 3." Request for Comment (RFC) 1939. http://www.internic.net/rfc/rfc1939.txt. 1996.

[RFC2045] Freed, N. and Borenstein, N. "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies." Request for Comment (RFC) 2045. http://www.internic.net/rfc/rfc2045.txt. 1996.

[RFC2046] Freed, N. and Borenstein, N. "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types." Request for Comment (RFC) 2046. http://www.internic.net/rfc/rfc2046.txt. 1996.

[RICHMOND 97] Richmond, Alan and Richmond, Lucy. "HyperText Transfer Protocol." http://WWW.Stars.com/Internet/Protocols/HTTP/article.html. The Web Developer's Virtual Library. 1997.

[SHAH 96] Shah, Rawn. "Beginner's JavaScript." http://www.javaworld.com/javaworld/jw-03-1996/jw-03-javascript.intro.html. JavaWorld. 1996.

[SHIFFMAN 97] Shiffman, Hank. "Making Sense of Java." http://reality.sgi.com/employees/shiffman_engr/Java-QA.html. Silicon Graphics, Inc. 1997.

[SPELMAN 97] Spelman, Jennifer and Rein, Lisa. "CSS or JSS: Which will better suit *your* needs?" http://www.netscapeworld.com/netscapeworld/nw-07-1997/nw-07-css.html. *NetscapeWorld.* 1997.

[SUN] *The Java™ Language: An Overview.* http://java.sun.com:80/docs/Overviews/java/java-overview-1.html. Sun Microsystems, Inc.

[VONDRAK 97] Vondrak, Cory. "Java." *Software Technology Review.* http://www.sei.cmu.edu/technology/str/descriptions/java_body.htm Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. 1997.

[WIUM LIE 96] Wium Lie, Håkon and Bos, Bert. *Cascading Style Sheets, level 1.* W3C Recommendation REC-CSS1-961217. http://www.w3.org/pub/WWW/TR/REC-CSS1. World Wide Web Consortium. 1996.

[ZGODZINSKI 97] Zgodzinski, David. "Will the Market or the W3C decide?" *Internet World.* Vol. 8, Num 6. Pg. 52. June 1997.