ECE 299S 2008 Server Design Report	
Team Members	Student Numbers
1. Rohaan Ahmed	995374385
2. Mohammad Hasanain Arshad	995357411
3. Heesham Ali	995266224
4. Raza Ansari	995632267
Instructor Name: Dario D. Degan	
Date Submitted: April 11, 2008	Number of Appendices: 5 (A to E)
Seminar Section: 0104	Lab Section: 0102

The ECE299 final project required the design team to create a fully functional, C++ based web server by appending onto an existing basic server, from ECE299 Labs 1-4, and adding several new features. At the beginning of the design stage, the team brainstormed and decided that the server design must be robust, reliable, fail safe, maintainable, expandable, intuitive, versatile and efficient.

The design process was undertaken using a combination of the spiral and the incremental design model. The server was designed in many separate modules after assessing the objectives, project plan, risk factors and the inter-dependence of features.

Testing was an integral part of the design process. A testing life cycle consisting of modular, integration, regression, white box, and system testing was put in place, and each module was taken through this testing phase.

The discovery of an error in the testing phase led directly to the debugging phase. In this phase, the precise source and location of the error was first detected, using output statements and debugging programs, and then fixed. The module was then taken back through the testing cycle.

After the final server had been implemented, the design team evaluated each module as well as the complete server with respect to the client requirements and the team's previously defined objectives. A successful performance by each individual module as well as the completed server during the extensive evaluation stage led the team to conclude that the server was a success. This conclusion was further validated by the fact that the team was able to address 100% of the project requirements as well as an additional 93.33% of the desired functionality that satisfied the various self-devised objectives.

© Rohaan Ahmed, 2009

The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder. All rights reserved.

1.0 Overview

The ECE299 Communication and Design course required our design team to utilize the understanding of software engineering design and practices gained throughout the semester and develop a Linux based, C++ programmed web server for a fictional client, employing the experience and knowledge gained from previous programming courses and ECE299 Labs 1 – 4. Apart from the basic server, created in the labs, the design requirements provided were flexible, which left most of the design decisions to the team.

In order to create a competitive and unique design, the team first derived the following list of stakeholders will be of importance during the server's life cycle.

- The design team
- The client (ECE299 course administration)
- The end user (will access the server through the internet)

Keeping these stakeholders in mind, the team then continued on to the design phase of the project.

2.0 Initial Design Decisions

At the beginning of the design process, the team had to choose a basic server from amongst the four servers built individually by each member during the labs. This basic server would allow the team a solid platform on which to build our more complex server during the design. In order to arrive upon the correct base server, the team first decided to create an objectives list. Apart from the initial decision making process, this list of objectives would also be useful throughout the design process as it will serve as a rubric with which the team would compare the server.

In order to come up with a strong list of objectives, the team employed two major decision making techniques, brainstorming and an objectives comparison chart. Free brainstorming was used to derive and sort an initial list of objectives during the first team meeting. The team then compared the objectives with the requirements of the project to shorten the list, which was then condensed further

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder.
All rights reserved.

using a silent brainstorming session. After the intensive brainstorming session, the team compared each individual objective to every other objective in a chart to obtain an objective ranking system.

After the extensive decision making process, the team decided that the following design objectives must be achieved during the design process of the web server (in no particular order).

- Robust Design: The server must be able to deal with most errors without external administration.
- Fail Safe Design: In case of an error or bug, the server should be able to inform the appropriate party (design team, administrator, client, user), and then resume normal service.
- Debuggable / Maintainable Design: The server should be designed in a way that would make it easier to debug and maintain.
- Flexible Design: The server should allow content and access flexibility to the client and the end user.
- Intuitive Design: The server should allow easy access to features and content for the client and the end user.
- Expandable Design: The server should be designed such that further expansion is always possible.
- Concurrent / Versatile Design: The server should be able to achieve multiple tasks of different types at the same time.
- Efficient Design: The server should perform tasks in a minimum amount of time with minimum loss of data.

Keeping these objectives in mind, the team then decided to integrate specifically well done features from each individual member's base server and create a robust platform for the large scale server [Appendix A – Brainstorming Map].

3.0 Component Breakdown & Project Plan

Before the team could begin the design of the server, an initial project plan had to be devised. In order to create a relevant project plan, the design team had to first decide the features it would implement into the server. The features to be

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder.
All rights reserved.

implemented, and their importance to the design, were decided based on the following criteria, as decided by the team during a brainstorming session:

I. Objectives: How well it meets the objectives initially set by the team.
II. Time: How long it would take to implement the feature.
III. Dependability Analysis: Dependence of the feature on other features
IV. Risk Assessment: How the feature's integration would affect other code that has already been implemented.

Depending on the above criteria, the team then implemented the web server features in the following order:

Stage I

1. Configuration File Format: Allows the client to easily configure the server to their liking.

```
InstallRoot:/home/ahmedroh/ece299/project/installation
DocRoot:/nfs/ugsparcs/a-c/a-c/ahmedroh/ECE299/Maverick/public_www
Port: 54092
```

Figure 3.1 - Sample Config File formatting and information (actual text).

2. Error, Access and Debug Log Files: Allows the design team and the client to easily view usage and error information and debugging logs.

```
192.168.4765 Sat Apr5 23:56:16 2008 get /index.html http/ 1.1 200 1985
```

Figure 3.2 - Sample Access log in the log file (actual text).

3. Error HTML Pages: Informs the end user of any errors.
4. Default Page: Displays a default web page for each domain.
5. Multiple Content Type Handling: Allows the client the ability to offer content type other than text files.
6. Support for Upper Case Requests: Enabled the server to work with upper and lower case client requests.

Stage II

1. Virtual Web Hosting: Allows the client to host multiple domain names.
2. Directory Content Listing: Informs the end user of the current directory's content.

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder.
All rights reserved.

3. Automatic Pathname Expansion: Allowed the end user to access other accounts.
4. Dynamic Content Handling: Allows the client to provide user specified dynamic content.
5. Concurrent Connection Capability: Allows multiple user connections simultaneously.
6. Graceful Server Shutdown: Allows the client to gracefully shut the server down.
7. Load Generation and Simulation: Allows the design team to test and debug the server as well as log performance statistics.
8. Clustering: Allows the client to set up the server at multiple stations and take advantage of a distributed server load setting.

Each feature was first studied for similarities and interdependencies and then implemented as a new module or as a part of an existing module [Appendix B – Component Breakdown].

4.0 System Overview

The proper functionality of the server depends on the successful execution of some specific steps. The server must first open a port to allow client transaction over the internet. Once a client request is received, the server binds with the client and opens a communication stream. The server handles the request depending on its validity. If a request is valid, the server responds by sending the requested content back to the client, however, if the request is invalid, the server sends back an error message to the client and resumes its regular operation.

The server handles incoming requests in the following manner [Appendix C – Software Flowchart]:

1. The server binds itself to a port by opening a socket (as specified in the configuration file).
2. The server recognizes an incoming connection and binds to the client.

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder.
All rights reserved.

3. The server calls the Parallel Thread library to create a new thread (instance) of the server (Netcom) object.
4. The server then receives an incoming request from the client through a newly created input/output stream.
5. The HTTPrequest module reads and parses the incoming request and authorizes its validity.
6. If the request is valid, using the previously created input/output stream, the server finds and sends the correct content back to the client.
7. Upon successful completion of transaction, the server keeps the connection with the client alive, until the user terminates the connection.
8. Along with the transactions, the server keeps track of the amount of open threads and the reception of the shutdown signal at all times. It also keeps a live log for all server activities.
9. If the server receives a terminate signal from the administrator, it waits until all the currently open connections have been serviced, and then gracefully shuts down.

5.0 Design Strategy & Decisions

One of the requirements set by the course administration was that the design must be implemented in the C++ programming language. There are many reasons for this, the main being that C++ allows for an object-oriented style of programming, which enables large scale design projects to be done in smaller pieces. To successfully implement as many features as possible while meeting all the objectives in a short time, the team also chose to follow a modular design approach. This enabled us to expand the design by developing newer object-oriented components separately and adding them to the larger, more complex server rather than adding to existing functions and classes which may have been developed by other team members, causing large scale conflicts.

Figure 5.1 - Sample code showcasing objects in one particular module.

```
Socket* sock = masterSocket->Accept();
Thread* thread = new Thread;
server = new netcom(sock, thread, networkbuffer);
```

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder.
All rights reserved.

Throughout the design process, the team employed a combination of two design models, the incremental model and the spiral model. The overall design process was incremental as each module was built by a different member of the team, tested, and then integrated into the server according to the project plan. Due to the time constraints, many modules were also implemented in parallel by different members of the design team, as in the spiral model. Once a module had been completed, it would undergo many steps within the testing cycle and then integrated into the entire system as a component. The simultaneous processes of development, testing and debugging allowed the team to concentrate on both implementing various features as well as achieving the design objectives.

6.0 Programming Optimization

One of the objectives that the design team had set out to achieve in the design of the server was efficiency. In particular, this means minimum runtime (due to minimum code complexity) and minimum memory usage. Apart from system efficiency, it is also important to write the server in a coding format that allows for easy understanding and expandability by other programmers. Therefore, the design team took special measures throughout the design process to achieve efficiency.

6.1 Use of Global and External Variables

To limit memory use, runtime and compile time, global variables were used to reduce the time it takes to create copy and pass variables throughout the program.

To allow multiple objects to use one variable, extern variables were employed, which limited memory use and runtime, but added to the compile time.

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder.
All rights reserved.

```
bool outputMessage (iostream, string URL, HTTPrequest object)
{
//Function responds back to the client on a valid request
        //access the requested directory and file
  string requestedFile = HTTPrequest-> Root directory + URL
}
```

Figure 6.1 - requestedFile, a global variable, is set using two local variables.

6.2 Use of Public Functions

To allow multiple classes to take advantage of common or similar functions, the design team implemented public functions within classes that can be accessed from anywhere within the program.

```
//A function of HTTPrequest is being accessed in Server.C (Main file)

 response = HTTPrequest->readAndParse(netstream);
```

Figure 6.2 - An example of a public function, readAndParse().

6.3 Derived Classes

To allow a new class to use an existing class' functions, variables and implementation, the team used derived classes wherever possible. These classes could be derived from within an existing library or a previously custom created class.

```
//HTTPmessage is a derived class of HTTPrequest


class HTTPrequest :public HTTPmessage
{// HTTPrequest Class…
…};
```

Figure 6.3 - The class HTTPmessage is derived from the class HTTPrequest

7.0 Testing

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder.
All rights reserved.

The adoption of the spiral model into our incremental design process increased the risk of conflicts between different modules of the server. To reduce the chances of errors and bugs, the design team devised an extensive testing procedure which had each module undergo a complete testing cycle [Appendix D - Testing Flowchart]. The following are the various phases of the testing phase:

1. Modular Testing: The designer of each module tested the module separately, concentrating on only the features implemented within the module.
2. Integration Testing: The designer of the module would integrate it with all the basic server and test specifically for errors within the newer module.
3. Regression Testing: Upon successfully passing the first two phases, the module would then be integrated into the larger server. In this phase, the designer concentrates heavily on the corner cases, trying to cause the newly created module or the server to crash.
4. White Box Testing: In this phase, the design team would test the newly integrated module as part of the entire system, testing each path through the server individually.
5. System Testing: In this black box testing phase, the design team tests the server by inputting random requests and matching the correct outputs.

The team used a web browser and the telnet command to connect to the server directly for each testing phase. This allowed each individual tester to test for unique cases, concentrating on the functionality of the new feature as well as the features already implemented. In order to automate the white and black box testing phases, the design team used an external program, called the load generator, to simulate two important components of the runtime functionality of the server.

1. Variable sized load (domain) files.
2. Simulation of incoming connections.

8.0 Debugging

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder.
All rights reserved.

If an error or bug was encountered in any of the testing phases, the designer of the faulty module took it through a pre-planned debugging cycle. The debugging cycle consisted of 2 phases:

1. Locating the error.
2. Resolving the error.

In order to locate an error, the designer employed one of two strategies.

1. Step-by-step Output Statements: The designer placed ‘cout’ statements in relevant places within the code to determine which loop case or function was causing the problem. This method also helped him determine whether or not the path the program takes through the module is the correct one. This method is most useful for quick debugging as it does not require too much time or effort.

```
cout << "New Thread Created\n";
int response = httpreq->readAndParse(netstream);
cout << "Exits ReadAndParse and enters server.C\n";
```

Figure 9.1 - Using ‘cout’ statements to determine the control flow path.

2. Debugging Program: If the step-by-step output method were unsuccessful in locating the bug, the designer would then use a debugging program, such as the Data Display Debugger or the GNU Debugger. These programs are especially designed to display the internal workings of the program and the memory to allow the designer to see exactly what the computer system is doing at each step within the program. This method is very extensive and was, therefore, always successful in locating a bug.

Once the bug was located, the designer would take it back through the development cycle, fixing any and all sources of errors within the code. One this is done, the module would once again go through all the testing processes until a final, working version is completed.

9.0 Design Assessment

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder.
All rights reserved.

The design team evaluated and assessed the functionality of the server at various points during the design phase. In particular, two types of evaluation processes were undertaken.

9.1 Component Analysis

Component analysis was mostly undertaken during the design process, upon the successful completion of the testing cycle by each component. The following components were evaluated in detail [Appendix E – Feature Models].

9.1.1 Configuration Files

The configuration file's functionality was tested by using several versions of the configuration file and testing the server to ensure it re-configures itself according to the specifications provided.

The evaluation result for this component was positive as the server met the desired specifications each time, and output an error message each time the configuration file was defective.

9.1.2 Load Files

The load files were tested by sending random requests to the server and checking whether or not the correct outputs were written into the test text files. This feature performed well under testing and as the correct output text was visible in the files.

9.1.3 Concurrency

The server's concurrency was tested by using both web browsers and telnet. Multiple simultaneous requests were made from the server, and the correct output behavior was monitored.

This feature was quite troublesome since it was first implemented. When tested in conjunction with other features, such as graceful server shutdown, it often crashed the program. However, all the problems related to this module were later resolved by devoting extra time to this feature and taking it through the testing cycle twice.

9.1.4 Handling Other Content Type

This feature was tested by requesting several different types of content from the server and studying the output.

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder.
All rights reserved.

This module performed well as it was able to handle all the content types the team had implemented, and output an error when it encountered a type that was not.

9.1.5 Load Generator

To test this external module the team tested it in two phases:

1. The program was ordered to generate multiple files of different sizes, filled with random text.
2. The program was ordered to delete all the files it had created in a certain directory.

The program performed perfectly in both cases, generating the correct load files as well as deleting them when ordered.

9.1.6 Graceful Shutdown:

This feature was evaluated by ordering the server to shut down while it was still in use by multiple connections. An output statement was used to test whether or not the shutdown signal was received.

This feature did not perform very well when concurrency was first implemented. However, after much testing, debugging and modifications, it all bugs were fixed and the feature performed perfectly, i.e., it received the shutdown signal, did not allow any new connections and shutdown as soon as the last connection was terminated.

9.1.7 Automatic Pathname Expansion

This feature was evaluated by ordering the server to access files from the directories of various other users.

This feature performed well under testing by allowing the user to access other directories and outputting an error when they did not have permission.

9.1.8 Dynamic Content Handling

This feature was evaluated by ordering the server to run a specified program and checking to see if the correct output was produced.

The feature performed well under all test conditions.

9.1.9 Directory Listing

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder.
All rights reserved.

This feature was tested by creating ‘dummy’ files in a directory and then requesting the server to display the listing for that directory.
The feature performed well under all test cases; however, its correct operation is heavy dependant on a working Configuration file.

9.2 Complete System Analysis

Upon the completion of the development and testing processes, the design team undertook a final server evaluation. This objective of this evaluation was to compare the final design of the server to our initial requirements and objectives as well as the comparison to the various stakeholder needs.

9.2.1 Requirement Based Analysis

At the beginning of the design project, the course administration had set up a few requirements for every design team while leaving most of the design decisions up to the team. Our server was designed to meet all these requirements from the beginning.

Requirement	Server
C++ based	Meets well
Object oriented	Meets well
Linux based	Meets well

Figure 9.1 - Mandatory course requirements and how the meets them.

9.2.2 Objectives & Functions Based Analysis

As stated in 2.0 Initial Design Decisions, the design team developed a list of objectives derived after much brainstorming and debate. The server, right from the beginning, was designed to meet these specific objectives along with the mandatory functionality specified by the course administrators. Upon the completion of the design phase, the team compared the final server’s features and functionality with our initial objectives list to determine our success rate.

Features Implemented	Objectives Met
Configuration File Format	Flexible, Intuitive, Maintainable
Log files	Maintainable, Intuitive, Efficient
Graceful Server Shutdown	Robust, Intuitive, Fail Safe

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder.
All rights reserved.

Error HTML Page	Intuitive, Robust, Fail Safe
Default Page	Intuitive
Upper-case Characters in URL	Intuitive, Versatile, Flexible
Handling Other Content Types	Versatile, Expandable
Virtual Web Hosting	Versatile, Expandable
Automatic Pathname Expansion	Flexible, Versatile
Listing Directory Contents	Intuitive, Maintainable, Efficient
Supporting Dynamic GET Requests	Versatile, Flexible
Adding Support for POST Requests	Versatile, Flexible
Concurrency	Versatile, Robust, Flexible, Efficient
Load Generator	Robust, Intuitive

Figure 9.2 - List of program features and their contributions to the objectives.

9.2.3 Risk Analysis

One feature the design team chose not to implement is clustering. The decision to not implement this feature came after a risk analysis session conducted during an emergency team meeting. During the meeting it was decided that it was in the favor of the team to pursue robustness in the concurrency feature as clustering is heavily dependant on concurrency. Also, the need to implement a graceful server shutdown mechanism, which became very complicated after concurrency had been implemented, was realized to be far greater than the need for distributed server load. For these reasons, and the fact that clustering would take an additional two days to implement, the team decided to cancel its implementation.

10.0 Evaluation of Success

Considering that we were able to implement 14 out of the 15 features we had set out to accomplish, and that each feature served to meet all our initial objectives as well as add quality to the server, the design team has concluded that the design was a tremendous success. By the end of the design, development and testing phases, we were able to accomplish 100% of our requirements [Figure 9.1], 100% of mandatory functionality and 93.33% of our objectives within the allotted time [Figure 9.2].

11.0 Conclusion

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder.
All rights reserved.

The final design of the web server was developed using an incremental-spiral process, addressing the various needs of the stakeholders. The functionality assessment of the server showed that it was robust, reliable, fail safe, easy to debug and efficient. In addition, the object-oriented nature of the server made it flexible and easy to expand in various usage situations. In the end, the design team was able to accomplish all the major goals set out at the beginning of the design process. This server met all the requirements that were listed in the project requirement statement as well as all the mandatory functions and the team's objectives.

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder.
All rights reserved.

Appendix A – Brainstorming Mind Map

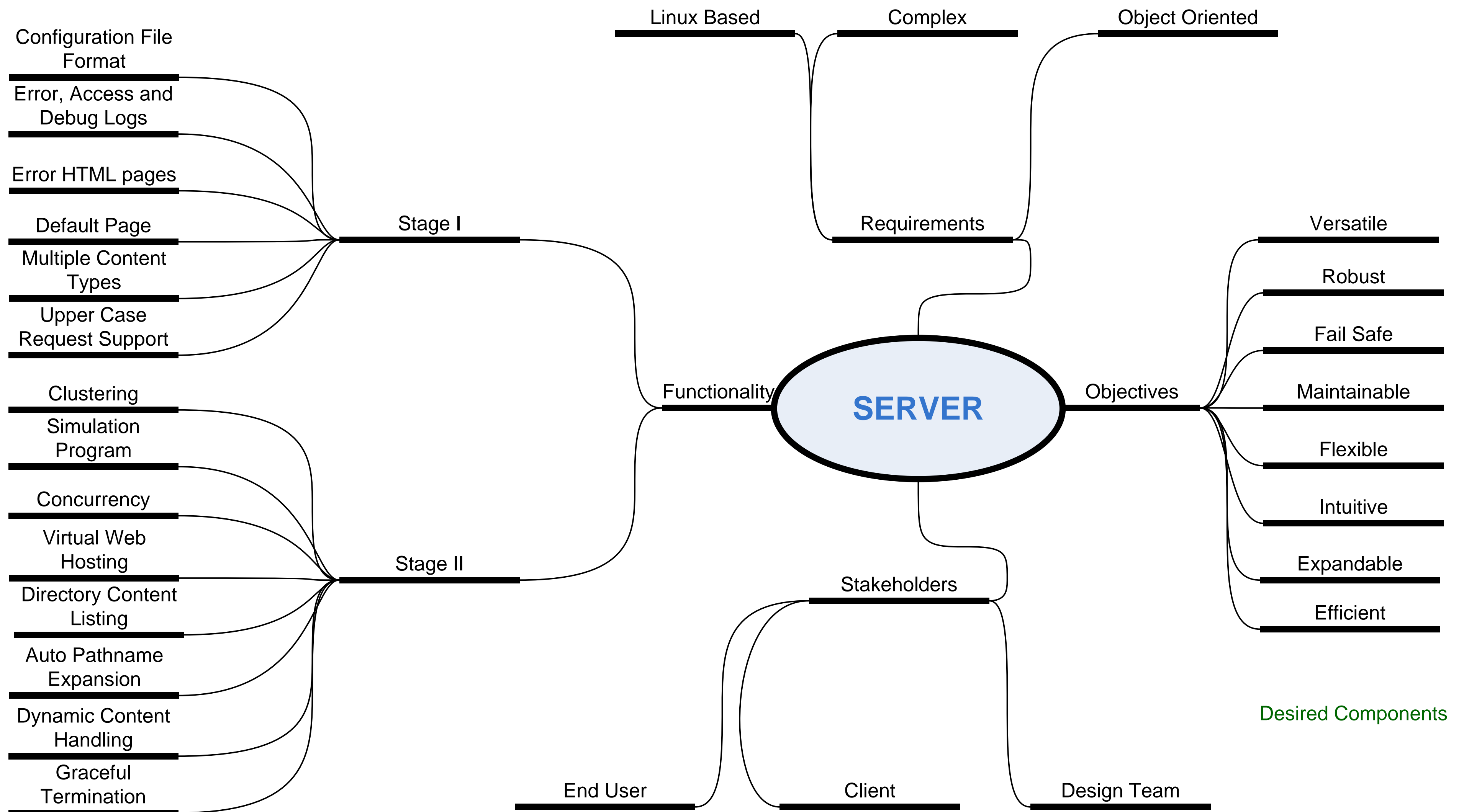


Figure A1

This brainstorming mind map was used extensively by the team during the design process.
The map identifies all the stakeholders and presents the final requirements, functionalities and (self-derived) objectives.

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder. All rights reserved.

Appendix B – Component Breakdown

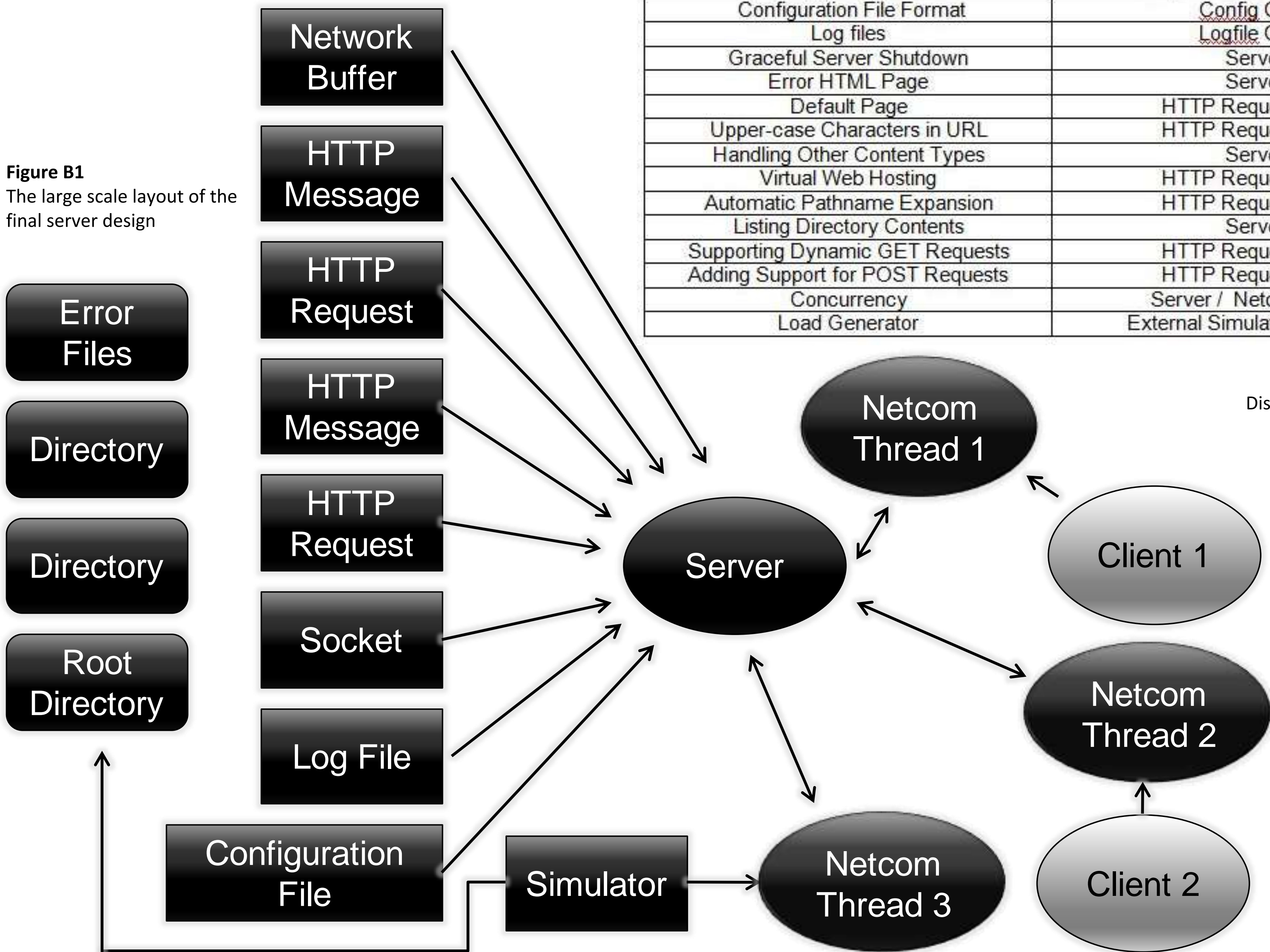


Figure B1
The large scale layout of the final server design

Stage II Feature	Implementation Module
Configuration File Format	Config Class
Log files	Logfile Class
Graceful Server Shutdown	Server
Error HTML Page	Server
Default Page	HTTP Request Class
Upper-case Characters in URL	HTTP Request Class
Handling Other Content Types	Server
Virtual Web Hosting	HTTP Request Class
Automatic Pathname Expansion	HTTP Request Class
Listing Directory Contents	Server
Supporting Dynamic GET Requests	HTTP Request Class
Adding Support for POST Requests	HTTP Request Class
Concurrency	Server / Netcom Class
Load Generator	External Simulation Program

Table B1
Displays in which module each feature was implemented

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder. All rights reserved.

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder. All rights reserved.

Appendix C – Software Flowchart

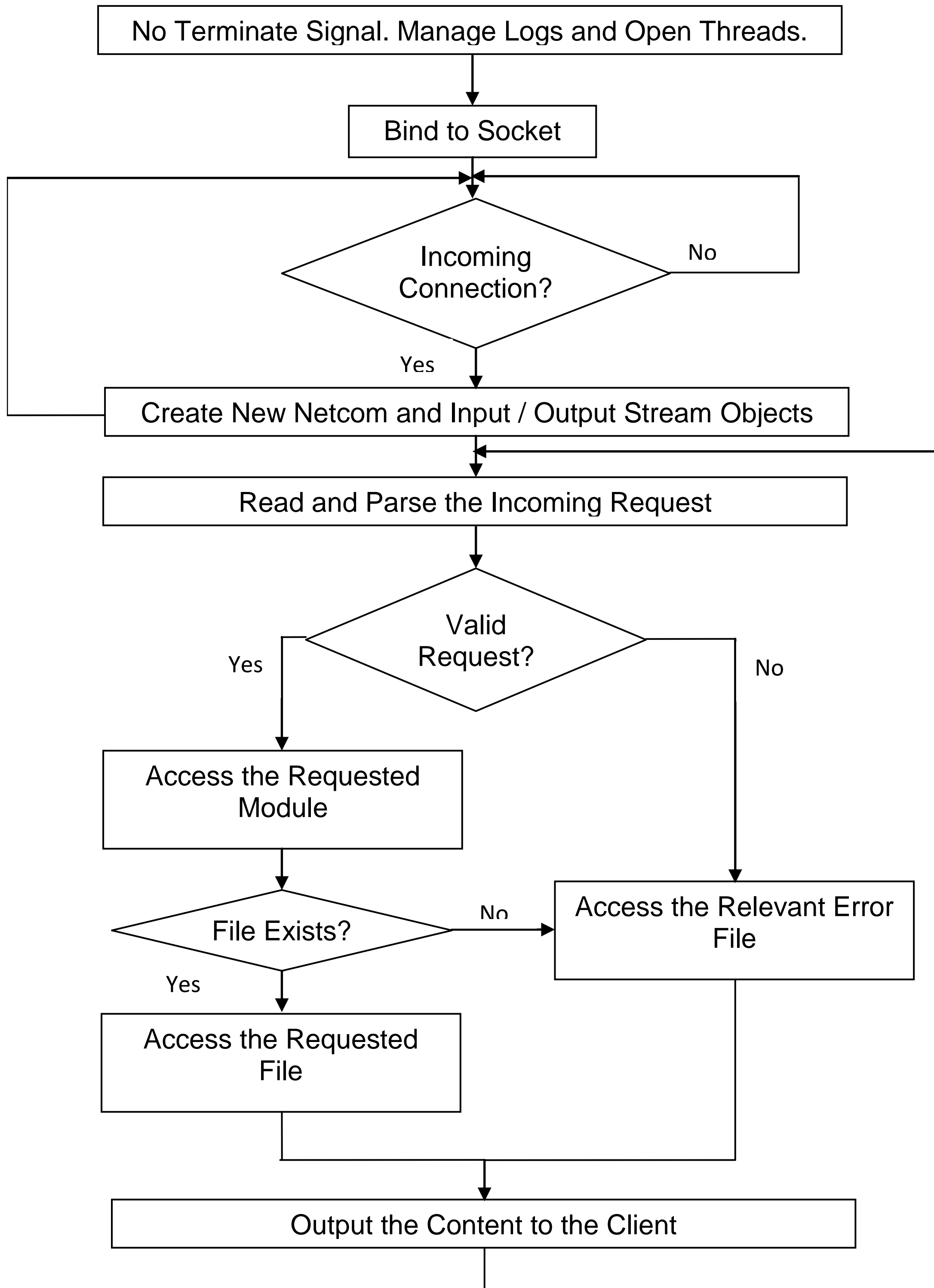
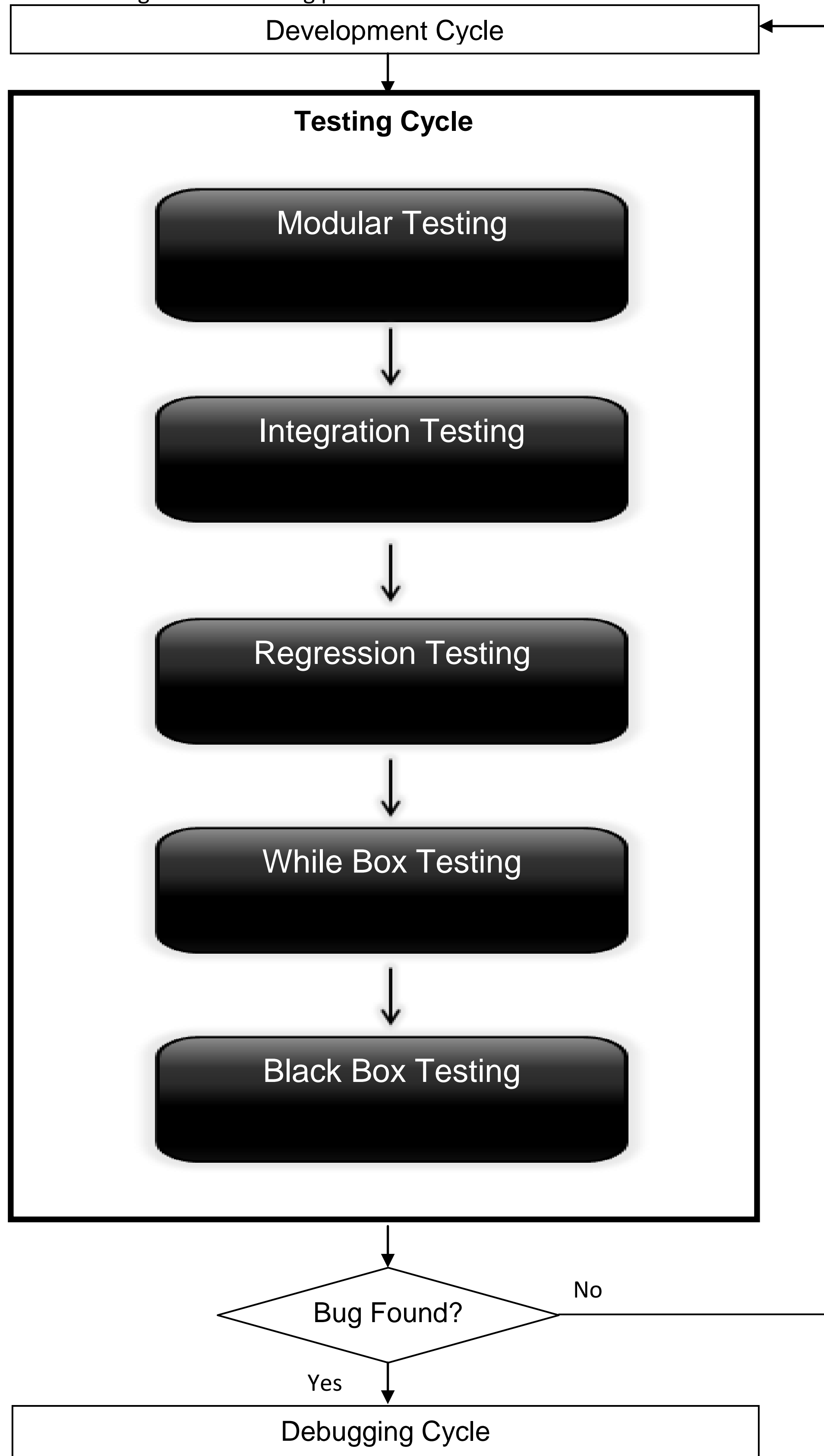


Figure C1 – Software Flowchart
This flowchart displays the overall path through the server when it is connected to a client

© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder. All rights reserved.

Appendix D – Testing Flowchart

Figure D1
Outlines the various stages of the testing phase.



© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder. All rights reserved.

Appendix E – Feature Models

Figure E1 - Configuration File
The server administrator configures the configuration file to change the behaviour / characteristics of the server. When the server starts, it reconfigures itself according to the instructions inside the configuration file.

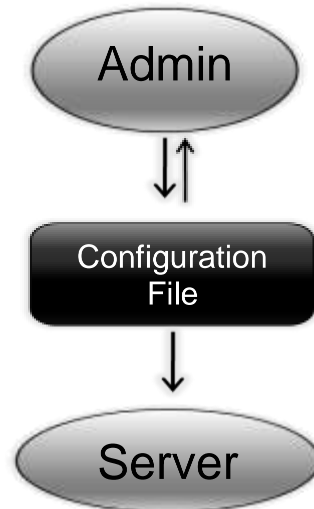
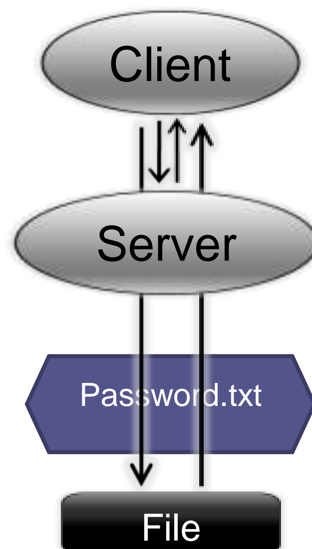


Figure E2 - Automatic Pathname Expansion
When a client requests a file contained within a different folder / user account, the server first prompts for a password from the Password.txt file. If the correct password is entered, the server accesses the file from the directory and relays it to the client.



© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder.
All rights reserved.

Figure E3 - Dynamic Content Handling
When the client requests dynamic content and provides the correct input parameters, the server triggers an external program that generates the dynamic files from the inputs. The server then sends the dynamic files back to the client.

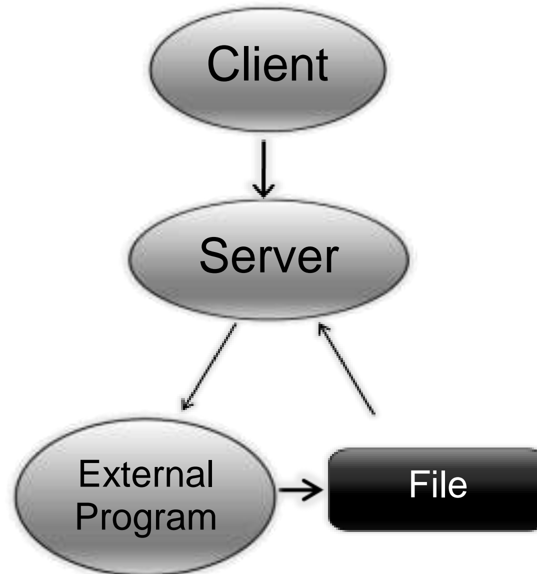
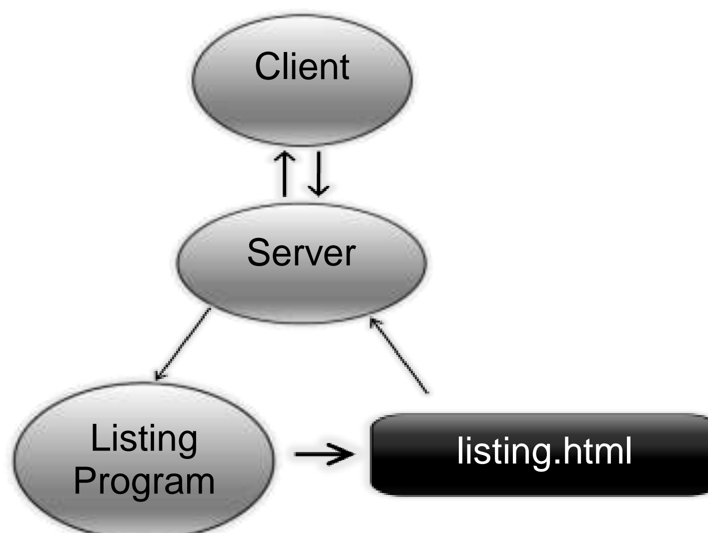


Figure E4 – Directory Content Listing
When the client asks for a listing of the directory content, the server runs an internal listing program that generates an up-to-date listing.html file within that directory. The server then outputs this file back to the client.



© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder.
All rights reserved.

Figure E5 – Load Generator
The load generator is a simulation program that is run locally. It can be used to do the following functions:

1. Create custom load filed with random data.
2. Simulate incoming connections and requests.
3. Test the server.

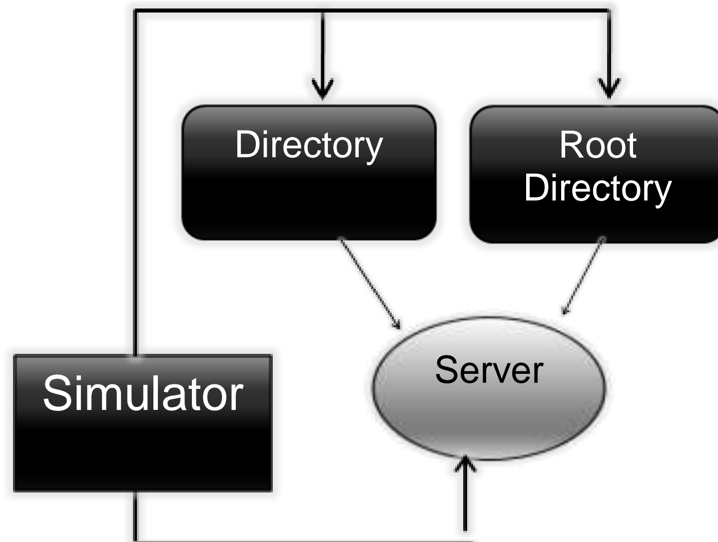
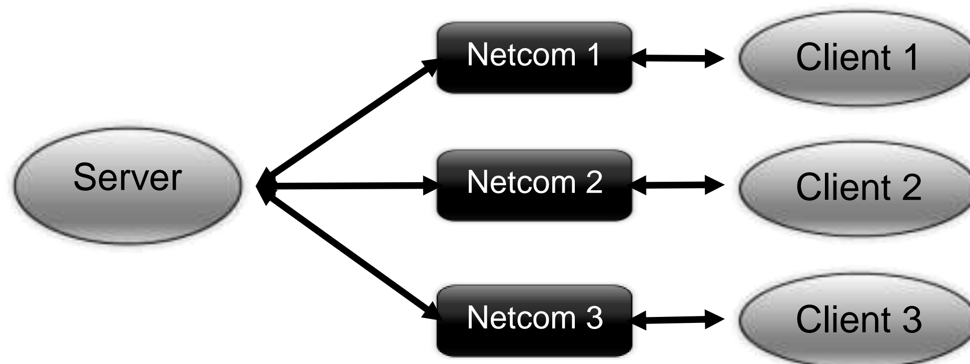


Figure E6 – Multithreading
Anytime a new user tries to connect with the server, it creates a new Netcom object thread dedicated to that one connection. The server grants all the client's requests through this object until a connection is closed, in which case, the server terminates the thread.



© Rohaan Ahmed, 2009
The contents of this document may not be reproduced or distributed in any way, shape of form without the prior consent of the author(s) and the Copyright holder.
All rights reserved.