1.1.1 Function Point Analysis

1.1.1.1 Inputs

The inputs for our project come from the graphical user interface and the communicator daemon. In the GUI, we have five tabs and a services tree that we use. The services tree is of high complexity because it dynamically changes to tell the state of your current services. The ``Application'' tab is of low complexity as it merely serves as an application manager for the service modules. Internal to that are the application modules, which we will state have a medium level of complexity, as they must take into consideration much more configurations that are individually tailored to specific modules. The ``IP'' tab is of low complexity because most all you should have to do is input the desired IP address. The ``User'' tab is of medium complexity as it is used to configure the passwords and options for all user accounts. The ``Status'' tab gives you log output from various applications and statistics from the operating system, hence it is of medium complexity. As a whole, the Java GUI has two TCP interfaces, one with another computer's GUI interface, and one with the UNIX communicator daemon. These are of low complexity, as it is very easy to communicate through sockets in Java. Finally, the inputs into the UNIX communicator are of medium complexity due to the nature of the development platform. In total, that tallies as nine separate inputs, one of high complexity, four of medium complexity, and four of low complexity.

1.1.1.2 Outputs

We have six main outputs that our project uses. We output to our own log, our ``Status'' tab, our services tree, and our three TCP interfaces. The log

should be of low complexity as it would just be a file that we append simple messages to. The ``Status'' tab will be a medium complexity factor because there will be a little work formatting the messages and displaying them on the screen. The services tree will be of high complexity because it must know the state of all running services and change dynamically as their status changes. Finally, the TCP interfaces that interface in Java will be of low complexity and the one in the UNIX layer will be of medium complexity, for the same reasons as stated above. In summation, there are six outputs, one of high complexity, two of medium complexity, and three of low complexity.

1.1.1.3 Queries

Our project has seven queries that it will use. Basically our queries are anything from which we can derive statistics. All of the queries are very easy to use, due to the ease of UNIX signal calls, TCP communication in Java and UNIX, and the ease of reading files. Therefore all of our queries are of low complexity. We can get the storage space available, CPU utilization, memory utilization, and network usage statistics from the OS layer via the proc file system. We can use Unix signals to determine information about the state of our running application services. We can also use TCP to communicate with the lower level of our application and with the other unit's GUI. Hence, we have a total of seven queries of low complexity.

1.1.1.4 Files

There are approximately twenty-four files that our project is going to use. They vary in complexity from low to high. The high complexity files are SMTP configuration, IMAP configuration, Samba configuration, and FTP configuration.

In most cases entire open source projects have been built around configuring these files. We realize just how complex editing these files can be, so we intend to only change a few key attributes of each. Even with our simplified modifications we still expect using these files to be of high complexity. Our own configuration file for the cluster will probably be of medium complexity. Considering we will be designing this file ourselves we will be aware of each option and what it does. There will also be a security file of our own design. It will be of medium complexity because it we will have to use encryption to create the password hashes we store in the file. Finally the low complexity files are the files we read OS information such as CPU, memory, and LAN interface usage from which reside in the proc file system, and our own log file which we append messages to. The remainder of the files are related to the installation package, and these range from pictures to applications. We shall currently assume that there will be eight files of high complexity, eight of medium complexity, and eight of low complexity.

1.1.1.5 Interfaces

Our project has five different interfaces. Our project will have to interface with the OS, the communicator daemon, the GUI, the application module framework, and our configuration files. The OS interface and the application interfaces will be handled by UNIX signals and the proc file system, hence they are of low complexity. The LAN interfaces will also be handled through the OS but these interfaces will be used to transfer our heartbeat and network file mirroring information. For these reasons we have determined the LAN interfaces to be of medium complexity. The GUI will be designed by us and should be of

medium complexity. Finally the application module framework and configuration files will also be of low complexity, due to the strict object-oriented approach we have taken towards our design architecture. Therefore there are two interfaces of medium complexity and three that are of low complexity.

1.1.1.6 Function Point Diagram

	Low	Medium	High	Total
Inputs	1 Î 3	4 Î 4	4 Î 6	43
Outputs	1 Î 4	2 Î 5	3 Î 7	35
Queries	7 Î 3	0 Î 4	0 Î 6	21
Files	8 Î 7	8 Î 10	8 Î 15	256
Interfaces	3 Î 5	2 Î 7	0 Î 10	29
TOTAL	**99**	**120**	**165**	**384**

Table 1: Function point matrix

1.1.2 AFP and COCOMO Analysis

1.1.2.1 Adjustment Factors

Factor	Description	Influence
Backup and Recovery	The system is a backup and recovery system, this element is critical.	5
Data Communications	The system depends on communications.	5
Distributed Functions	The system is a fail-over cluster. Hence, distributed functions are not critical.	1
Performance	Performance is more of an issue for the services (applications) that are run, but the fail-over system cannot be a performance problem.	2
Operational Environment	The environment should be fairly generic.	1
On-line data entry	There is little data to be entered – and Java is the planned language for the screens – so this is of medium importance.	3
Multiple Screens for Input	One screen with multiple tabs to simplify the user interface.	1
On-line Update	There is no data to be updated – so this is not important.	1
Interface Complexity	On average, this project has interfaces that are of average complexity.	3
Re-usability	The API will be designed for maximum re-usability.	1
Process Complexity	The process is already well defined.	1
Installation Ease	This is one of the key goals for the project.	5

Table 2: Function point adjustment factors

Multiple Sites	The system will be designed so it can be installed at multiple sites for one customer (and at multiple customers' sites).	4
Ease of Use	This is another key goal for the project.	5
TOTAL		38

Adjusted Function Points (AFP) = 384 Î (0.65 + (0.01 Î 38))

AFP = 395.52

Here we use 34 as the language factor because it is the weighted average of the languages we will be using (25% C, 5% UNIX scripts, 70% Java).

Lines of Code (LOC) = 34 Î 395.52

LOC = 13,448

Finally we use the organic model with basic COCOMO analysis as that model best describes our project.

Labor Month (effort) = 2.4 Î ((LOC / 1000)^1.05)

Labor Month = 2.4 Î (13.448^1.05)

Labor Month = 36.7

Develop Time (schedule time) = 2.5 Î ((Labor Month)^0.38)

Develop Time = 2.5 Î (36.7)^0.38

Develop Time = 2.5 Î (3.9338)

Develop Time = 9.8

People Needed = Labor Month / Develop Time

People Needed = 36.7 / 9.8

People Needed = 4