CS 486 – Capstone Project

Functional Specifications

(Revision 1.0)

Submitted to

Dr. Doerry

By

Team Fugu:

Erik Wilson

Ben Atkin

Nauman Qureshi

Thad Boyd

On

March 1, 2004

Table of Contents

1. Introduction

1.1. Executive Summary

On the 5th of January 2004 the United States Geological Survey (USGS) approached Northern Arizona University's (NAU) Capstone Project with an idea of developing OS tools for OpenBSD. Team FUGU (*http://www.cet.nau.edu/~fugu/*) was formed in order to develop this project. USGS is a world leader in the natural sciences through their scientific excellence and responsiveness to society's needs. The USGS Astrogeology Program uses OpenBSD due to its renowned security but the costs in time for installation and maintenance of the operating system is a big drawback.

On the 16th of January 2004 Team FUGU was chosen to develop an automated installer and an automated patcher for OpenBSD which when developed would alleviate the USGS of their problems. A requirements document was accepted on the 18th of February. This document provides a detailed Functional Specification for the proposed software solution.

1.2. Problem Description

The USGS serves the Nation by providing reliable scientific information to:

- Describe and understand the Earth.
- Minimize loss of life and property from natural disasters.
- Manage water, biological, energy, and mineral resources.
- Enhance and protect our quality of life.

The Astrogeology Research Program is located at the Flagstaff Field Center and contains a team of over 80 research scientists, cartographers, computer scientists, administrative staff, students, contractors, and volunteers working to support the efforts to explore, map, and understand our solar system. Fields of particular interest to them are mapping, planetary geologic processes, remote sensing and monitoring, and scientific analysis, which lead to answers about our neighboring planets. Throughout the years, the program has participated in processing and analyzing data from various missions to the planetary bodies in our solar system, assisting in finding potential landing sites for exploration vehicles, mapping our neighboring planets and their moons, and conducting research to better understand the origins, evolutions, and geologic processes operating on these bodies.

The Information Technology (IT) department of the Flagstaff Field Center has relied on, among other operating systems, OpenBSD to provide the critical network services necessary to run their systems. OpenBSD serves the needs of the USGS, but installation and maintenance of the operating system is a huge cost in terms of time. The creation of our proposed installation and maintenance tools will greatly benefit the USGS as well as the entire OpenBSD community as a whole by saving time for the administrators which use the OpenBSD operating system.

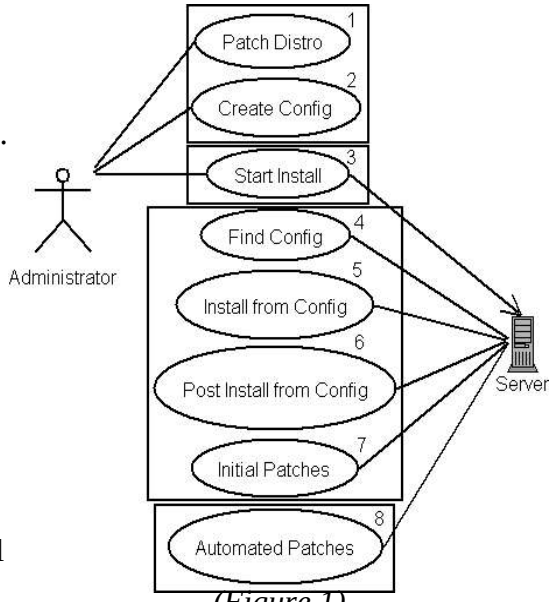
The central requirements of the software are as follows:

- An automated installer will be developed to replace the tedious manual installation process currently used by OpenBSD.
- An automated patcher will also be developed to keep the operating system up to date and free of security holes. This function will free the administrator from patching each computer individually.

The following sections of this document describe our proposed solution and specific requirements in greater detail.

1.3. Product Description

As indicated in *Figure 1*, our proposed automated installation and patching processes involves the following steps:



(Figure 1)

1. Modify the current manual installation and upgrade system to create the auto-installer.
2. For a class of machines the system administrator will create a configuration file.
3. The system administrator boots the new distribution and there will be approximately a five-second time-out before the automated installation begins (during this time the administrator can drop to a console or perform an interactive installation).
4. If the automated installation isn't interrupted, the installer will search for the configuration file from a variety of locations, and then parse that file.
5. From this configuration file the installer will be able to partition the disk, format the partitions, configure networking, and install packages.
6. The automated installer will run a post-install script that is obtained from the configuration file.
7. Initial patches will occur and the system will reboot into the new OS.
8. The patcher will be a regularly scheduled task, it will be run both as a *cron* job and using */etc/rc* scripts, to ensure updates will not be missed if a computer is shut off. It will be highly configurable, allowing for either standard source patches from the OpenBSD FTP site (and mirrors), or custom binary patches from the local network.

The previous steps when implemented comprise our proposed automated process, which is derived from the current manual process. Please read on to the next section, *Product Functions*, for more details as to what capabilities the product will perform.

1.4. Product Functions

The following two subsections introduce the major functional requirements for the automated installer and the automated patcher. For further details please see *Functional Specifications* in *Section 3*.

1.4.1. Automated Installer

Upon the completion of the automated installer, it will fulfill the following lists of tasks:

1. Must be substituted into existing OpenBSD installation environment.
2. Must be able to handle either clean install or upgrade an existing OS installation.
3. Must read an installation configuration from multiple sources.
4. Must be able to handle all the installation steps currently performed by the manual installation process.
5. The tool must also handle pre and post installation scripts.

1.4.2. Automated Patcher

Upon the completion of the automated patcher, it will fulfill the following lists of tasks:

1. Arguments can either be passed in on the command line or set in a configuration file.
2. It must handle both source and binary patches.
3. Binary patches must be able to run a pre-install, post-install, pre-uninstall and post-uninstall scripts, and contain install and uninstall processes.
4. The patch system must keep track of what patches have been installed.
5. The system must be able to send an email, using the standard UNIX mail system, to the system administrator(s).
6. A highly desirable feature is that the tool be able to run in the installation environment.

1.5. General Constraints

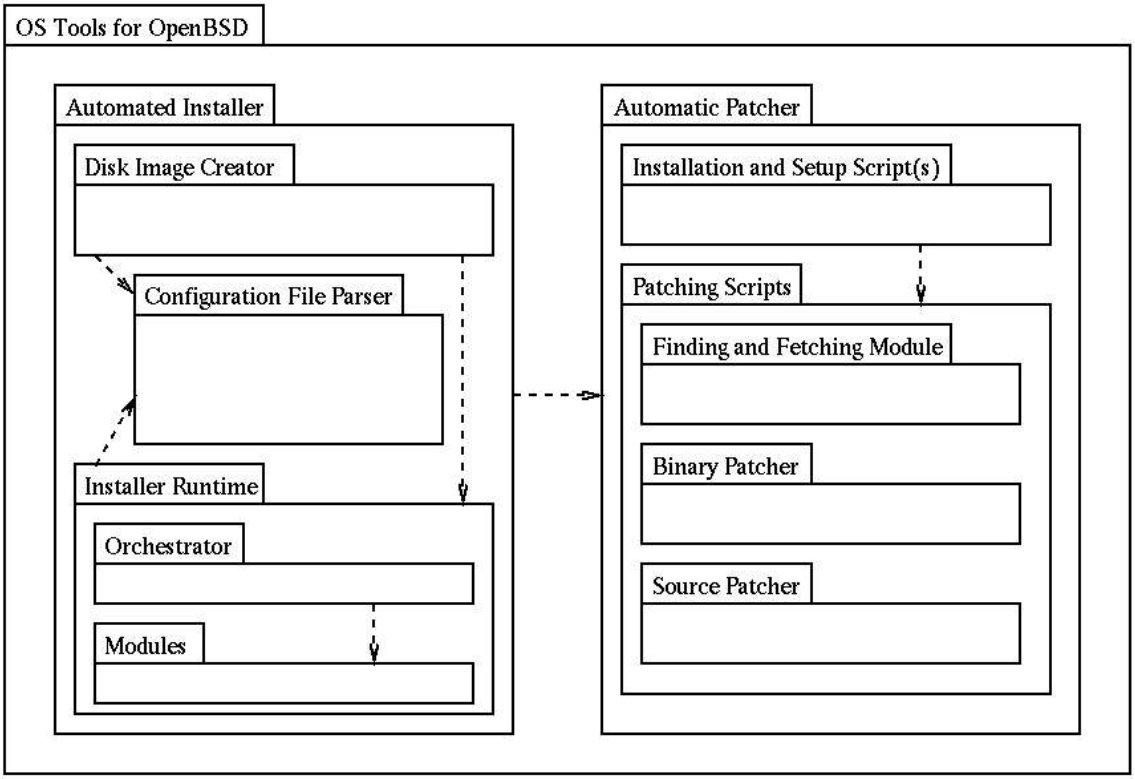
The USGS has placed very few constraints on the development of our tools for OpenBSD. Through a series of meetings we have been clearly told what the requirements of the software are and those are the only constraints set upon Team FUGU. We are open to use any language for the software's development. OpenBSD can run on almost any kind of hardware, therefore any hardware constraints are not applicable to our project.

1.6. Assumptions

Team Fugu is well equipped both hardware wise and skill wise. Our client offered us any kind of hardware that we might need on the way to the development of our proposed software solution and this has been accompanied by the assistance of the IT department at the College of Engineering and Technology (CET). Team Fugu is fortunate in that the Flagstaff Field Center is close enough to have frequent meetings and the availability of our sponsor to resolve any potential issues when they arise.

2. Software Architecture Overview

This section contains a high-level description of our software product's architecture, by organizing the design into packages. A package diagram is shown below as *Figure 2*:



(Figure 2: Package Diagram)

The following subsections contain a description of each package in the above figure, in top-down order.

2.1. OS Tools for OpenBSD

We are designing and creating two *OS Tools* for our client to use with *OpenBSD*, an Automated Installer and an Automatic Patcher. While these tools are largely independent, they interact in that the installer will be able to apply security patches, getting the system "up to date" before it is booted into and system services are started.

2.2. Automated Installer

The *Automated Installer* is similar to RedHat Kickstart and Solaris Jumpstart in that after the configuration file and boot media are properly set up and put into place on the system, a full install will be made on bootup without any user input. Hard disk partitions are set up, hardware is set up, software packages are installed, services such as ssh and print daemons are set up, and any pre-install or post-install scripts specified in the configuration file are run at the appropriate time.

2.3. Disk Image Creator

At the core of the installer is the *Disk Image Creator*, which parses and validates the configuration file, and creates a bootable disk image for a given architecture (i386, Sparc, or Sparc64) that can be written to floppy, CD, or other boot media. It is likely that some features of the Disk Image Creator will only work on OpenBSD (such as building from source), but most will work on any type of UNIX workstation.

2.4. Installer Runtime

On the boot media generated by the Disk Image Creator will be an *Installer Runtime* that we create, based on the code for the Interactive Install from the OpenBSD website. The Installer Runtime will use an *Orchestrator* script, that will determine which *Modules* are run, and in what order. The modules will include such things as setting up the network and setting up partitions, and anything else that can be done by the Interactive Install.

2.5. Automatic Patcher

The *Automatic Patcher*, our second tool, will have an Installation and Setup (maintenance) script, and a script which fetches and installs binary and source patches. The patching tool will be based on *Tepatche*, which provides part of the tools that our client needs, but the interface will be redesigned to suit our client's needs, and new features, especially binary patching, will be added.

2.6. Installation and Setup Script

The *Installation and Setup script* will install the patcher to disk, and add the *Patching Scripts* to the OpenBSD boot scripts and into *cron*, a task scheduling tool that comes with OpenBSD and most other UNIX systems. The frequency at which the tool shall be run will be specified in a configuration file in the installation directory or on the command line.

2.7. Patching Scripts

The *Patching Scripts* are comprised of three modules, a *Finding and Fetching Module*, a *Binary Patcher*, and a *Source Patcher*. The Patching Script can be invoked by the user at any time, and will be invoked at regular intervals by *cron*. The script can also be run on each system boot, and by the installer, ensuring that the system is no more than a day out of date at any given time

2.8. Runtime Order

The *Finding and Fetching Module* will be run first, and will find patches on the servers specified in the configuration file and will fetch only the ones which match the criteria specified in the configuration file. The Patching Script will run the Binary Patcher or Source Patcher, as appropriate, for each patch fetched and stored on the disk.

The *Binary Patcher* will apply the patch to a directory, and run a script provided with the patch.

The *Source Patcher* will unpack the patch to at temporary directory, set up configuration files, and run the appropriate make files to build and install the compiled source to the directory.

The *Patching Script* will update the information in the packaging system after the patch is complete. If a patch fails, an error will be sent to a log file and/or an email address, as specified in the configuration file.

3. Functional Specifications

Functional specifications are given for both the automated installer as well as the automated patching system. Details on the description, input, process, and output are listed for each specification.

3.1. Automated Installer

The automated installer consists of a disk image creator, a configuration parser, and installer runtime. These sections are as follows:

3.1.1. Disk Image Creator

The disk image creator is a command-line utility, written in a scripting language. It accepts a number of arguments, and has an accompanying man page.

1. **Description:** The disk image creator will provide options for building floppy disk images, ISO CD images, and images for booting off the network. This fulfills requirements *1* and *4* from *Section 4.2.1* of the Requirements document.
   Input: The current source tree for OpenBSD will be used, as well as patch modifications for that source..
   Process: The source code will be modified using the patch facilities, and then compiled using the existing Makefiles.
   Output: Upon compilation there will exist all of the standard installation media, including but not limited to: ISO file, floppy disk images, and ramdisk file.

2. **Description:** The disk image creator will provide an option for validating a configuration file. This fulfills a postmortem requirement not listed in the original Requirements document.
   Input: The configuration file will be included for the patch process detailed in the previous requirement.
   Process: A predetermined grammar will be used to validate all sections, keys, and syntax, see the next section, *Configuration Parser*, for more details.
   Output: If the configuration file does not parse into the predetermined grammar an error will occur notifying the user of the problem and disk image creation will cease, otherwise a success message will be produced.

3.1.2. Configuration Parser

The configuration file parser is responsible for determining that the configuration file is in the correct format and will be correctly parsed during the automated installer process. The parser fulfills a postmortem requirement therefore these specifications do not correspond to any particular requirements in the Requirements document.

1. **Description:** The parser will be run whenever information is requested from a configuration file by the installer runtime.
   Input: A configuration file name and path will be provided as a command line argument for the parser, as well as optional section and key arguments.
   Process: A static grammar within the parser will determine if the sections, keys, and syntax are used correctly. This will most likely be accomplished by using regular expressions facilities found in *grep* or *awk.* If a section and key is provided the parser will additionally locate the associated value.
   Output: Depending on the command line arguments given to the parser it will either return a success or failure code, or the value for a given section and key argument.

2. **Description:** The user may invoke the parser on the command line to validate any configuration files which are created.
   Input: A configuration file name and path will be provided as a command line argument for the parser
   Process: The process used will be identical to that of the previous specification.
   Output: The parser will return a verbose success or failure code to allow the user to identify where the error occurred within the configuration file.

3.1.3. Installer Runtime

When the system is booted, the installer will begin with a five-second timeout, where the user is given the option to drop to a console or perform a manual installation.

1. **Description:** The installer will be capable of performing either an installation or upgrade of the operating system based on configurations from multiple sources. This fulfills requirements *2* and *3* from *Section 4.2.1* in the Requirements document.
   Input: Configuration files will be gathered in the following precedence:
   1) Start by checking the CDROM and floppy for network configuration.
   2) Attempt to locate configuration files defined in the previous precedence for general system configuration.
   3) Attempt to locate machine specific configuration files based on IP address or host name from CDROM, floppy, disk, and network.
   Process: The installer runtime will call subroutines with arguments obtained from the configuration files which have been parsed by the configuration parser.
   Output: The installer will report the progress of the installation on the console (standard output). See S*pecification 3* of this section for further output options.

2. **Description:** The installer runtime will run pre and post installation scripts capable of copying files from hard drives. This fulfills requirement *5* in *Section 4.2.1* as well as requirement *6* in *Section 4.2.2* of the Requirements document.
   Input: The configuration files listed in *Specification 1* of this section will provide pre and post installation scripts and the current functionality of running *install.site* and *upgrade.site* scripts will be maintained.
   Process: The configuration parser will be used to extract the pre or post installation scripts from the configuration files. Hard drives will be mounted prior to the pre-installation script being executed in order to facilitate copying files from hard drives.
   Output: the results of executing the scripts will be displayed on the standard output, or possibly by a method described in the next specification.

3. **Description:** Options will be provided in the configuration file to generate a logfile based on the standard output, and to e-mail this logfile to the system administrator (if email has been set up). This specification fulfills a postmortem requirement not listed in the Requirements document.
   Input: The standard output created from S*pecifications 1 and 2* of this section will comprise the log file, information from the configuration files will be used to determine email settings.
   Process: The standard output of the installer runtime will be redirected through a pipe to a log file.
   Output: Output will be defined by configuration file, exists in either file form or sent through email.

3.2. Automated Patcher

As derived from the Requirements document the patching script will fulfill the following specifications:

1. **Description:** Tepatche will be modified using *ports* and *packages* to provide binary patch functionality. A single server for each desired architecture will host the binary patches once they are created. This fulfills requirements *1, 2, 3, and 4* in *Section 4.2.2* of the Requirements document.
   Input: The Tepatche configuration file or command-line input will be used to determine what actions the patcher will take (eg: use source or binary patches, if the computer should host binary patches). Additionally Tepatche will be modified to ensure it's configuration file can be parsed in the correct grammar, this is independent of the *Configuration Parser* listed in *Section 3.1.2*.
   Process: Tepatche will be modified to use the existing *ports* and *package* facilities to create, install, and uninstall binary patches. The *patch* facility is already utilized by Tepatche to install and uninstall source patches.
   Output: Tepatche will produce a patched system, or if the computer is designated as a binary server for an architecture it will additionally host the binary patches. Please see the next specification for more details on the output of the automated patcher.

2. **Description:** Logs will be kept for the automated patcher with several possible destinations and verbosity levels of the logging available. This fulfills requirement *5* in *Section 4.2.2* of the Requirements document.
   Input: The Teptache configuration file or command line arguments will be used to determine the verbosity of the desired log. These methods will also be used to ascertain if the logs should be sent via email to the administrator.
   Process: The standard output of the patching process may be redirected to a log file or a simple success/failure message may be created depending on the verbosity level desired.
   Output: The automated patcher will place a message on the system log, and a more detailed message in a program log file. Old log files will be rotated. The logs can be sent to a specified email address.

4. Use Cases

The following sections provide use cases to succinctly describe possible scenarios which might occur in order to use our product. Sections are provided for the installation and maintenance of OpenBSD, and subsections exist for each of those sections.

4.1. Installation of OpenBSD

The installation of OpenBSD involves two possible scenarios and their associated sections: the creation of disk images to be used for a class of systems and the process of booting that image onto a computer.

4.1.1. Running the Disk Image Creator

Scenario: Bob has been assigned a task by his manager, Ernest, to create an ISO image which contains the automated installation developed by Team Fugu to be used with a particular architecture.
Actors: Bob, an entry level IT intern, has rudimentary knowledge of the UNIX shell, and no knowledge of OpenBSD. Ernest is Bob's manager and he is an experienced system administrator, he is familiar with the local network configuration, and has an intermediate knowledge of OpenBSD.

User Steps and system responses:

1. After reading the instructions on Team Fugu's website, Bob downloads the disk image creator and uncompresses it into a directory.
   Explanation: Bob will use the *tar* utility to uncompress the file into a subdirectory within his home directory.
2. Ernest modifies the permissions of */usr/src* to allow Bob to write to that directory.
   Explanation: Only an administrator (root) can modify permissions of this directory, although the source could be defined to exist in a different directory this is the most beneficial location for it to live.

3. Bob begins the Disk Image Creator script.
   Explanation: The script downloads all of the required source packages, and uncompresses them into the /usr/src directory.

4. Bob is prompted for basic network configuration which the automated installation uses. Ernest in turn will supply any information which Bob can not provide.
   Explanation: The automated installation must know such things as: whether to use *dhcp* or *static* IP configuration, servers and the locations for additional configuration files.

5. The disk image creator places the network configuration file in the appropriate place and compiles the source, resulting in an ISO which contains the automated installation system.
   Explanation: The disk image creator patches the source code and uses the existing Makefile system to create a distribution.

4.1.2. Booting the Newly Created Disk Image

Scenario: Ernest needs to boot the image which Bob created in the previous scenario to install or upgrade OpenBSD on a computer.
Actors: Ernest is an experienced system administrator, he is familiar with the local network configuration, and has an intermediate knowledge of OpenBSD. He is also comfortable with a wide variety of text editors on the UNIX system and accustomed to editing configuration files.

User Steps and system responses:

1. Ernest reads the instructions listed on Team Fugu's website or in the README file for booting a disk image. He then records the image onto a CDROM.
   Explanation: There is a multitude of third party software available for recording ISO images onto a CDROM, Ernest has several such utilities available to him and will select the one which he prefers the most.

2. Ernest uses a text editor to modify example configuration files.
   Explanation: The example configuration files provide a comprehensive basis for all of the possible options definable for the automated installation.

3. After Ernest has created the configuration files for his systems he will run them through the configuration parser to ensure there are no errors in the files.
   Explanation: The configuration parser will ensure that the configuration file has no syntax errors, if there are any they will be verbosely described.

4. Once Ernest has ensured the configuration files are correct he will place these files in the network locations which he defined in the previous scenario.
   Explanation: The configuration files will be transferred to the location which was defined in the network configuration for the ISO image to be access via FTP, TFTP, HTTP, or even disk.

5. Ernest places the CDROM into the desired system and boots the computer from that CDROM. He can then take a break to get a cup of coffee.
   Explanation: Ernest may have to configure the BIOS or change the system's configuration to boot from a CDROM, but as an experienced system administrator he is familiar with this process. After the CDROM boots there will be no further interaction required from him so can leave to do something else.

6. The automated installation will prompt for user interaction then continue to locate and parse the configuration files. The system will continue to perform the installation or upgrade and once completed may email the administrator if configured to do so.
   Explanation: A more detailed explanation of the automated installation system can be found in the previous sections.

7. After the automated installation system has completed Ernest can return to retrieve the CDROM.
   Explanation: If the computer was setup to always try to boot from CDROM Ernest will have to remove it in order to boot the newly installed OpenBSD.

4.2. Maintenance of OpenBSD

Maintaining a newly installed or upgraded OpenBSD system can take on the following two scenarios: the automated patching has occurred without error, or an error occurred somehow in the process. These scenarios are listed in the subsections as follows:

4.2.1. Automated Patching has Occurred without Error

Scenario: Bob is in charge of receiving the email for the automated patching system. He checks his email regularly, at least once a day.

Actors: Bob, an entry level IT intern, has rudimentary knowledge of the UNIX shell, and no knowledge of OpenBSD**.** Ernest is Bob's manager and he is an experienced system administrator, he is familiar with the local network configuration, and has an intermediate knowledge of OpenBSD.

User Steps and system responses:

1. Bob starts up his favorite email client to check his daily email.
   Explanation: It is assumed the email for the automated patcher has been designated to be delivered to Bob, or Bob has access to the administrator (root) email account.

2. The success of a patch for each computer has been sent to Bob and the status is clearly depicted in the subject line for the email.
   Explanation: Both the computer host name and the patching status is clearly displayed in the subject line, eliminating the need for Bob to read the entire contents of the message.

3. On the unlikely occurrence of a failed patch Bob will alert his manager, Ernest, to the message.
   Explanation: Ernest will then perform the steps listed in the next scenario.

4.2.2. Automated Patching has Occurred with Error

Scenario: Ernest has been alerted by Bob as to an error which occurred during the patching process on some machine. He must analyze and fix the problem for the computer to remain secure.
Actors: Bob, an entry level IT intern, has rudimentary knowledge of the UNIX shell, and no knowledge of OpenBSD**.** Ernest is Bob's manager and he is an experienced system administrator, he is familiar with the local network configuration, and has an intermediate knowledge of OpenBSD.

User Steps and system responses:

1. Bob has forwarded Ernest an email created by the automated patching system which details the output of the attempted patch and the resulting error.
   Explanation: Ernest will immediately have knowledge as to which machine the patching failed on via the contents of the subject line.

2. Ernest will analyze the contents of the message to determine if it was an error with source or binary patching, which machine, what time, and the cause of the error.
   Explanation: For example: in the unlikely event that a partition is full this will be easy to determine from the contents of the failure email, any cause of error should be fairly easy to determine and therefore not time consuming.

3. As an experienced system administrator Ernest has several possibilities, however the first suggested action would be to run the patching system manually on the machine from command line.
   Explanation: If Ernest is unable to determine what the problem is immediately from the contents of the email then performing a patch manually may provide insight to the problem simply by watching to process run. If the patching error occurred using binary patches he should then try using a source patch.

4. If Ernest is still unable to determine the cause of the error as an experienced system administrator he must access any sources of information available to him in order to solve the problem.
   Explanation: The problems which occur during patching will be verbose enough to get Ernest started on a path towards a solution for the problem. In the very unlikely event that there is a problem with the patch itself he can always contact the OpenBSD forums for assistance.

5. External Interface Requirements

There are two parts to the interface, user interfaces and software interfaces. User interfaces includes all interactive portions, such as configuration and error reporting, and are detailed in *Section 5.1*, whereas software interfaces are those which will run automatically, and are detailed in *Section 5.2*.

5.1. User Interface

Due to the automated capabilities for our system the amount of desired user interface is as minimal as possible, however there are some user interfaces which must occur simply by convention. In particular these user interfaces are with the configuration file, user feedback, and some other tools that require user interface will be provided if time permits.

5.1.1. Configuration File

The primary user interaction with our program will be through a simple text configuration file. The file will include user specifications on data backup, partitioning, file locations, patching schedule, and any other necessary settings.

We will include a well-commented sample configuration file containing common settings, which will be easy for a sysadmin to modify.

5.1.2. User Feedback

Installation activities will be displayed on the screen and recorded in log files. Any installation errors will be visible in these formats.

Additionally, the patcher will report any errors by sending E-Mail to root and any other addresses specified in the config file.

5.1.3. Other Tools

If time permits, we may add graphical tools for such tasks as scheduling the patcher, generating the config file, and reporting overall progress of installation and patching.

5.2. Software Interfaces

There are several software interfaces which our proposed system must adhere to in order to function properly, they are described in the following subsections:

5.2.1. Launching the Installer

The installer will run similarly to the current version, but without the interactive prompts. The installer may be put in any bootable medium, such as a CD, floppy, hard disk partition,or network drive, and will run at boot time.

5.2.2. Partitioning Tools

In the case of a new installation (as specified in the config file), disks will be partitioned according to settings in the config file. In a format similar to fstab, the config file will specify the partitions, their locations, sizes (in B, KB, MB, GB, or percent of total disk size), file systems, and mount points.

5.2.3. Network Setup

Depending on setting in the configuration file, an IP will be assigned to the machine using DHCP or static assignment.

5.2.4. Remote Files

The OpenBSD installer only includes base files. Additional installation files, to be specified in the config file, must be transferred from other machines. The config file will specify an FTP or HTTP server, or an NFS location, from which these additional files may be retrieved.

5.2.5. RC Scripts for Patcher

The patcher configuration file will include the frequency at which to run the patcher and a set of valid servers to check for patches.

6. Performance Requirements

Performance Requirements are measurable user and computing performance characteristics. User performance characteristics are given for common tasks that will be completed using our product. Because our product consists only of run-once installation scripts and small, daily run patching scripts, computing performance isn't a concern. While the patching script can severely impact performance, it is the calling of build scripts, that are outside the scope of our project, that are a concern. Because of this, the only feasible solution is to educate our users about the performance advantage of binary patching. The performance requirements for the two tools are listed here:

6.1.Automated Installer

Four out of five entry-level system administrators should be able to start a customized automated installation within thirty minutes, without needing to consult any documentation other than what is provided with the system.

- They shall be able to instruct the installer to set up partitions, as desired.
- Their systems shall have the network, *sshd*, and the automatic patcher (which are all optional) working properly upon the first boot.
- They shall be able select which packages are being installed, and understand how to go about setting up custom packages.

6.2.Automated Patcher

The performance requirements for our automated patcher are as follows:

1. Four out of five entry-level system administrators shall be able to set up basic source patching, fetching the patch files from an OpenBSD web or FTP server, within ten minutes, without needing to consult any external references.
    - The first run of the patcher shall succeed.
2. Four out of five intermediate system administrators shall be able to set up customized binary patching, downloading pre-built patches from their own web or FTP server, within thirty minutes. Because this involves the administrator using multiple programs, needing use external references is acceptable.
    - The first run of the patcher shall succeed.
    - They shall be able to view a log that the patch install succeeded.
3. Three out of five system administrators, during the setup process, shall have found, in the documentation or printed on the command-line, a message warning about the performance implications of source patching.
    - The message shall suggest using binary patching on slower systems or systems where there is demand for high performance at all hours of the day.

7. Design Constraints

One of the main advantages of OpenBSD is that it is easily installed and can run on legacy systems. The main hardware constraints that are involved are as follows:

- Machines must have minimal of approximately 8 megabytes of memory to load ramdisk.
- A minimal of 1 gigabyte of hard disk capacity is preferred.
- Intel's i386 and SPARCs are the preferred computer architectures for this project.

USGS is operating on modern machines which leave less room for any hardware constraints regarding this project. Though the ramdisk installation method is preferred, there are other installation methods to circumvent the memory requirement such as installing from an uncompressed image on a partition. OpenBSD can run on Alphas, HP300 (and above), Intel's i386 (and above), and many other such architectures. This takes care of most of the hardware constraints. There are no other forms of constraints applicable other than the ones mentioned above.

8. Attributes

Once the automated installer and patcher are developed and running, all of the specified functions will be thoroughly tested to assure they run perfectly. As mentioned earlier, the function of the automated patcher will be to keep the systems up to date. Documentation will be provided in the 'man' pages and also in README files.

Along with the documentation, there will be referrals on the website such as the design document, and many other such helpful documents, from which the client can get help on the rare occurrence that they encounter a problem.