CS 486 – Capstone Project

Project Development Plan

(Version 1.1)

Submitted to

Dr. Doerry

By

Team Fugu:

Erik Wilson
Ben Atkin
Nauman Qureshi
Thad Boyd

On

February 16, 2004

Table of Contents

1. Introduction

The purpose of this Project Development Plan is to serve as a basis for all project development activities, particularly the early stages, by outlining how we plan to tackle the project and how we will manage the processes involved.

On the 5th of January 2004 the United States Geological Survey (USGS) submitted a project for a CS486 team to develop an installation tool kit and patch update system for the OpenBSD operating system. The USGS station located here in Flagstaff is involved primarily with image processing of data obtained from NASA spacecraft. To support their robust networking environment there are several dozen enterprise class servers, many of which run OpenBSD for its renowned security. The problem which USGS faces is the installation of the OpenBSD operating system on multiple machines, as well as the security maintenance of these machines in the form of patching. The time involved with these processes becomes greater as the number systems which are present increases; to solve this problem Team Fugu proposes to automate these tasks which otherwise would be chronologically inefficient to perform.

On the 28th of January 2004 USGS invited Team Fugu to further discuss the project aspects and requirements. A *Project Overview File* was given to each of the four members of Team Fugu – the document included sections on the project requirements, the project prerequisites, and other essential sections for the overview of the project. The following sections of this document are the details of the Project Development Plan, which have been constructed from team meetings, the *Project Overview File*, and our meeting with the USGS.

2. Team Members

Team Fugu is comprised of the following four members:

- **Erik Wilson** *(Team Leader / Organizer):*

Erik is the leader and responsible for team organization. Even though he is in charge his decisions can be overturned if there is a simple majority vote.

- **Ben Atkin** *(Communicator / Research):*

Ben's duty is to ensure that our group is communicating efficiently within our team and with outside contacts; he is also responsible for most of the preliminary research that occurs.

- **Nauman Qureshi** *(Recorder / Documentation):*

Nauman is in charge of keeping our team notebook, meeting minutes, and for quality assurance for any documentation that is created.

- **Thaddeus Boyd** *(Facilitator / Website Developer):*

If there is a dispute, Thad will attempt to resolve it. In addition he will update our website to include all of the documentation which is created during our project.

3. Deliverable Schedule

The following is a schedule of all major milestones and reviews required for the class as well as intra-team milestones and sponsor meetings (please refer to *Section 7* for dates related specifically to our development process):

[Requirements and Early Design]		
2/2	**Initial Research**	Initial research on CVS and shell scripting for OpenBSD installations will be completed.
2/11	**Meeting with Sponsor**	Detailed discussion of requirements.
2/18	**Requirements Document Preliminary Presentation**	Present Overview of project, sponsor invited.
2/20	**MILESTONE 1**	Functional specifications 75% completed.
2/25	**Final design Phase presentation within the team Meeting with Sponsor**	This is the final design that will be implemented and will be handed to the sponsor. Any changes needed to be made shall be made at this point.
2/28	**Functional Specification**	Final Draft to Sponsor
[Implementation and Testing]		
3/1	**Team Coding Standards Document**	Hard copy plus posted on team site
Week of 3/1	**Design Review Presentation I**	Covers functional specifications and final implementation schedule
3/3	**Meeting with Sponsor**	Solidify any architecture discrepancies.
3/15	**Software Design Specification**	Describes the architecture behind your implementation, including all classes, methods, etc.
3/23	**MILESTONE 2 Meeting with Sponsor**	Implementing phase to be completed by 50%. Display current progress.
4/4	**Design Review Presentation II**	Covers evolution of requirements, current implementation status, and plans for completion
4/7	**Usability/Functional Testing Plan and materials**	
4/14	**Meeting with Sponsor**	Display functionality of product.
4/28	**Capstone Presentation Dry runs**	
4/30	**Capstone Conference Final Project Presentation**	The big one!
5/3	**Team Notebook Team Website**	Final, detailed, comprehensive review and evaluation.
5/5	**Final Report**	Complete Final Report on the project. Must include usability and functional validation results.

4. Risk Analysis

The following is a condensed list of possible risks which are applicable to the project, ranked by severity and probability, each described and containing a strategy for mitigation:

Risk	Description	Probability	Severity	Mitigation Strategy
1. Does not perform to standards	Tools do not perform their tasks correctly. For example, installer does not partition disk correctly or patching system does not update correctly.	Low	Critical	Test thoroughly. Make absolutely sure that programs perform to specifications.
2. Interface unusable	Tools are inconvenient and actually increase operation time rather than save it.	Low	Critical	Test early and often. Communicate with sponsor and send test versions for feedback.
3. Cannot back up existing files	Installation procedure is only suitable for completely wiping systems and is not equipped to back up important files (SSH keys, etc.) where necessary	Medium-low	High	Begin research on this as soon as possible and ask sponsor for specifics. This should be a trivial bit of coding as long as we know which files we need to look for and back up.
4. Programs not fully automated	Tools save operation time but still require user input.	Low	High	Write code to operate automatically first and worry about accepting user input later.
5. Limited or no hands-on functionality	Tools are completely automated and do not allow user input.	Low	Medium	Write automatic code first as it is the priority, but code expecting to add user input overrides and to make doing so as easy as possible.

5. Requirements

The following sections comprise a plan for discovering all of the requirements associated with this project. ***Note:*** UML examples have not been provided for brevity,

please consult a *UML Distilled* text for specific examples.

5.1. Capture of Primary Requirements

A thorough description will be provided for each requirement and these descriptions will be supported through UML Use Case Diagrams; the combination of these will be used to capture the primary requirements. These diagrams will be included in the Requirements deliverable and may be referred to if needed in tracking requirements to implementation.

5.2. Showing Derived Requirements

Similarly, derived requirements will be shown through textual descriptions and use case generalization, i.e.: the capturing of alternative scenarios by extending an existing use case. They will be differentiated from the primary requirements by notating such in the textual descriptions.

5.3. Tracking Requirements to the Architecture

Tracking requirements to the architecture will be achieved through UML Package Diagrams. The package diagrams will show packages of classes and the interactions among them, thereby creating a broad overview of the architecture. These package diagrams may be linked to particular requirements through the textual description when necessary.

5.4. Tracking Requirements to the Design

Tracking requirements to design will be accomplished through UML Class Diagrams. The class diagrams will be kept mostly to the conceptual level to avoid being bogged down in implementation details.

5.5. Tracking Requirements to Implementation

Tracking requirements to implementation will be provided by using UML State Diagrams in the context of a particular use case or object. If there is more than one object collaborating or several applicable use cases then a UML Activity Diagram may be used instead.

5.6. Testing for Requirement Coverage

Testing for requirement coverage will be provided by philosophical team discussions and rigorous testing cycles. Although each team member is required to test his own code thoroughly there may be entire team meetings dedicated towards this task at the end of a development phase.

6. Design Documentation

There are several documents we can expect to deliver to both the sponsor and mentor which have not been included in the list of deliverables, these are:

- **Configuration file format** for installation process
- **Sample output** of an automated install run
- **Log file format** for patch update system
- **User Manual** for patch update system (this may be provided by command line)

In addition, it is also possible that preliminary architecture and design diagrams may be provided during team meetings with the mentor to ensure that the correct methodologies and approaches are being used.

7. Development Process

We would like to make the system as complete and usable as possible, so we will give our sponsor the possibility to preview our tools multiple times before it is complete. Because there is an interactive component the Waterfall method is too rigid and not completely applicable. Therefore we will use the Scrum development process (http://www.controlchaos.com/) which allows for a minimum of three sprints during our allotted time. Daily scrum meetings are unfortunately unfeasible; however, a minimum of three hour-long meetings per week is feasible, and a backlog will be maintained by the recorder. Therefore, the following is a list of preliminary scrum sprints with expected deliverables:

[Scrum Sprints]		
2/28	**Specification Delivered**	Skeleton implementation available
3/28	**Implementation 50%**	Major functional requirements complete
4/28	**Final Product**	All requirements done including performance and usability

8. Software Configuration

8.1. Language

We plan to primarily use shell scripting and Perl for our tools. At the time of this writing, we are still researching shell scripting methods and how best to approach the problem.

8.2. IDE

No standard IDE will be determined for the whole team. Rather, each team member is encouraged to play to his own strengths and find an IDE, or just a text editor, that suits him best. Team members are encouraged to share their thoughts on and successes with various IDE and text editors in the hope that each team member can benefit from the others' experiences and find the most suitable editor for his own use - compatibility issues are eliminated as source files are plain-text.

8.3. Version Control

We will use CVS (Concurrent Versions System) to handle different versions of our code. This software allows easy rollback if a mistake is found in a newer version of code.

8.4. Integration

We will meet at least three times a week and discuss our code, showing examples where appropriate and offering suggestions on how best to make every individual piece of code most suitable for integration. If, during the integration phase, the integrating coder does not understand the specifics of a piece of code, he is to meet with the original coder to help clarify. Finally, it is important to note that there are two major portions of this project, the automated installer and the automated patching system, and that these two major portions do not need to be integrated.

8.5. Development and Licensing

As this is an open-source project, we intend to build on existing code where appropriate. This primarily includes the existing OpenBSD installation tools, but may include any other pieces of open-source software we deem appropriate. Our primary consideration for another piece of software to modify is Tepatche, an existing OpenBSD automated patching system recommended by sponsor Ernest Bowman-Cisneros.

9. Hardware and Platforms

9.1. Architecture

Our software will be designed to work on Intel i386 and Sun Sparc machines. Our research indicates that the difference in the platforms should already be built into the existing OpenBSD tools we are modifying, so little or no code should have to be written differently to facilitate differences in the platforms. There are no minimal requirements per se for these machines; this is because there alternatives to using the ram-disk installation method which would require approximately 8 megabytes of memory for installation.

9.2. Specific Machines

The team has already acquired a Sparc computer for testing purposes. Each individual member is also expected to have an i386-based machine to use for test purposes. At the time of this writing, some team members are currently seeking to borrow extra Sparc or i386 machines for home testing purposes. Both Dr. Doerry and USGS have offered to let team members borrow hardware for testing purposes and at this point it is merely a matter of ironing out the details.

9.3. Utility Software

We will primarily use various standard *nix-based programs for this project. We use FTP for file transfers, SSH for remote logins, OpenOffice/StarOffice and Microsoft Word for documents, and, as previously stated, whatever text editors we find suitable for coding purposes. So far the only code we have written is the XHTML and CSS for our website, which has been written in Kate and Vim. Other software may be used at team members' discretion as the need arises.

10. Client Communication

We will send an E-Mail to our sponsors at USGS on minimally a weekly basis notifying them of the week's progress, needs, and concerns. We will meet with the sponsors in person or by conference call whenever it is deemed useful. Additionally, all deliverables will be given to the sponsor, and when possible a meeting will be held before they are due to give the client an opportunity to appropriately review on them.