Design

Objectives

The goal of this assignment is to give you experience in:

(1) Effectively applying software design principles
(2) Effectively applying architecture and design patterns
(3) Using UML to document a software design
(4) Designing in a team environment

Assignment

With the PRD, Functional Specification, and User Manual in hand, you have a relatively complete description of FCS's *external* behavior. In this assignment we move our focus from *what* FCS does to *how* it does it.

The assignment is to do a design of your FCS system's *internal* implementation, and to document your design in a formal Design Document. The primary audience for your Design Document is the team of engineers who will have responsibility for implementing the software (in this class that's you, but in other situations it might be somebody else). Your design should specify all of the important aspects of the system's structure and operation in enough detail that someone other than you could implement the software based on your Design Document and produce a system that is true to your creative vision. If you don't include enough detail, the implementers will be forced to make most of the real design decisions, and the resulting system will probably be very different from what you had in mind. If you include too much detail, you might as well skip the design and write the code. Your design should specify the system's overall architecture, the major abstractions in the system, and how the abstractions work together to implement each of the FCS features.

While doing your design, you should focus on applying the software design principles that were discussed in class, and look for opportunities to apply the architecture patterns and design patterns that we've studied. Our evaluation of your design will be influenced by how well and how frequently you apply these principles and patterns.

You should use whatever written explanations, UML diagrams, and anything else that you think will be effective at communicating your design. The goal is to communicate effectively and efficiently, not to kill trees.

Your Design Document should include an architectural overview of the system. Beyond that, it should include at least the following:

(1) Class diagram(s) that specify the abstractions in the system. Specific details on class attributes and operations should be included, although less-important or obvious details may be left to the implementers. Relationships between classes should be clearly shown. You might have one big class diagram, or you could have one class diagram for each subsystem or group of related classes, whatever you find to be more effective.

(2) Sequence diagrams and/or Communication diagrams for each of the FCS use cases. The implementers will need detailed explanations of how the system should implement each of the use cases.
(3) State diagrams for any classes that go through interesting state changes during their lifetimes. (E.g., Would it be useful to draw state diagrams for Files?, Directories?, Repositories?, etc.)
(4) Activity diagrams for interesting business processes. (E.g., Would it be useful to draw activity diagrams depicting how end-users use FCS to perform their jobs?)
(5) Activity diagrams for interesting algorithms. (E.g., What's the algorithm for determining if a file is allowed to be removed from the repository? Is it complicated enough to justify drawing an activity diagram?).

These diagrams should be accompanied by written descriptions or anything else that enhances communication with the reader.

Traceability

Your design should demonstrate traceability to the PRD and Functional Specification, meaning that your design should account for everything mentioned in those documents, and it shouldn't add features that aren't mentioned in those documents.

Testability

Your design should take testability into account. It should be easy to write automated unit tests for the different pieces of the system, with the possible exception of the user interface, which may be difficult to test in an automated fashion. Writing automated unit tests will be part of a later assignment, so plan ahead for it now in your design.

Application Paper

Write a short paper that explains specifically how you applied architecture patterns, design patterns, and good software design principles in your design. Which patterns did you use? Where were they used? How does your design support testing? Etc. This paper will help those evaluating your design to better appreciate its elegance and beauty (or lack thereof). Again, killing trees isn't the goal, communication is.

Updated User Manual

Update your user manual to reflect any changes you've made to the program's operation since you initially wrote the manual. You should also incorporate feedback received from the TA on the first draft of your manual.

Deliverables

Four hard copies of your Design Document, Application Paper, and updated User Manual (one for me, and one for each member of the quality assurance team)

Grading

Your design will be evaluated both by the TA and your quality assurance team. Your grade will be assigned by the TA, but the quality assurance team's report will be taken into account.

Your design will be evaluated on the following criteria:

1) Effective communication with the reader
2) Traceability to the PRD and Functional Specification
3) Effective application of good software design principles
4) Effective application of architecture and design patterns
5) Effective use of UML diagrams
6) Testability

You are encouraged to meet with the TA as your design progresses to get early feedback and suggestions. Your design will be held to a high standard, so use your time wisely, and do all that you can to improve your design before it's due.