Software Engineering Homework #4 – due February 21, 2008

Instructions

The following pages are a design and planning document template. Please follow these directions for filling out this template carefully. Items in italics are information that you are to fill in, beginning with the project name, date, and version number. Use the same version numbering system that we used with the requirements document.

If any section is not applicable, put N/A in the body of the section (do not delete the section). Also, if the information has already been given in another document, refer to that document.

Each team must send a copy of this completed document to barrett@cs.nyu.edu by midnight on the due date. The subject line of the email message should contain only the word "hw4".

Project Name
Design and Planning Document
Date
Version *major.minor*

1 Document Revision History

Rev. 1.0 *date* - initial version

2 System Architecture

Here you should describe the high-level architecture of your system: the major subsystems and how they fit together. Provide a UML class diagram of the subsystems which includes the name of each subsystem and its public interface. Also indicate using association relationships which subsystems make use of which other subsystems. The class diagram should correspond to a regular or abstract class in your code. Also, wherever possible items should be linked back to your specification. Ideally, you can match up everything in the specification with where it is implemented in the design.

3 Class Design

In this section, describe how each subsystem is implemented with component classes. Again, use a UML class diagram to show the different classes and their relationships with each other. You should clearly show how the public interfaces described in the system architecture are implemented in terms of the classes making up the subsystem.

4 Detailed Design

In this section, go into those important facets that are not described in the class diagrams above, such as descriptions of critical algorithms, protocols, and key invariants.

5 Testing plan

In this section, give a brief description of how you plan to test the system. Thought should be given to how mostly automatic testing can be carried out, so as to maximize the limited number of human hours you will have for testing your system. The purpose of this

section is a reality check that your design will in fact be testable with a reasonable effort. You should specifically discuss design decisions that affect testing, and describe any test interfaces built into the system in this section. You must discuss here your plans for unit testing (approach, when are they created, when are they run), integration testing (what order), system testing (what kind), regression testing (how are you going to organize and run them).

6 Plan

From your design you should prepare a plan. In particular, you should list all of the tasks to be done, a preliminary assignment of tasks to people in the group, estimates of the time for each task, dependencies between tasks, and a preliminary division into builds (e.g., which features are implemented in the first build, second build, and so on). The plan should be designed to get some prototype system running as quickly as possible and then growing towards the full project over a sequence of builds. Please pay extra attention to the dependency relationships between tasks; you will almost certainly run into the situation where one task isn't done and everything else is waiting for it. Part of the purpose of the plan is to try to avoid such situations as much as possible.

7 Checklist

The following checklist is provided to help you think about whether the document is complete and correct. These questions will also be used by the team which reviews your design. You may want to add your own additional questions to the list.

7.1 Content

- Is the design complete, well-organized, and clear? Are the components and interfaces specified in enough detail that the design could be turned over to an independent group for implementation and still be understood?
- Is the design specified hierarchically?
- Are diagrams used effectively?
- Are design patterns used effectively?
- Is the level of abstraction appropriate and consistent?
- Is information hiding used effectively to isolate complexity?

- Is information hiding used effectively to isolate parts of the program that might change?
- Is the design minimal?
- Is coupling between components in the design minimal?
- Is the strategy for the user interface design covered?
- If needed, is the database design specified?
- Does the design include a coherent error-handling strategy?
- Are all the requirements in the requirements document covered by the design in a sensible way, by neither too many nor too few building blocks?

7.2 Testing

- Does the design avoid nondeterministic behavior?
- Does the design include self-checks?
- Is the design stateless wherever possible?
- Does the design include a test interface?
- Are all the testing stages (unit, integration, validation) covered?
- Is the regression test strategy covered?
- Are non-functional validation tests necessary and covered (stress, performance, security, etc.)?
- Does the testing plan include walkthroughs or inspections?

7.3 Planning

- Does the plan include multiple iterations, each adding functionality incrementally?
- Does the plan take dependencies among different components into account?
- As much as possible, does the plan avoid situations in which the entire team would have to wait if one task is not finished on time?
- Is the timeline realistic?