CSE 403
Autumn 2013

Design (Phase 2b)
Due Friday, November 1, 2013

Objective

Produce a detailed architectural design of your system, (ideally) before you implement your ideas in code. A clear design becomes a blueprint for implementation, akin to following a recipe. A common name for this document is a System Design Specification (SDS). Among other things, your SDS should answer the questions: what are your major parts (modules, packages, classes, or other units of abstraction), what are the responsibilities of each part, and how do the parts interact/collaborate?

Background

Note that the SDS in this course will be treated as a *living document*. Therefore, you will be asked to provide updates to it at periodic points in the development cycle. For ease of editing and distribution, this document must be in electronic form.

As part of developing the SDS, you may need to revise your SRS (requirements document). Keeping these documents up to date with changes as they happen will make it easier to keep your team and customers in sync, and to meet your later deliverables. (Side note, exploring ways to alleviate the currently error-prone manual labor necessary to keep these documents current is the subject of **traceability** research in software engineering.)

The SDS provides a detailed definition of the system's software components. It identifies the major **modules** and their functionality, and the **interfaces** between modules, required to implement the system. It should address the design of the system from the customer's viewpoint, *as well as* that of the developer or administrator. (This may require separate perspectives and/or diagrams. The customer's perspective may be more abstract and less technical.)

Assignment

Provide a document that includes the following information:

- High-level architecture diagram and rationale
- Design decision rationale
- Data organization description
- UML class diagram of detailed design

High-level architecture pattern diagram and rationale:
Identify at least one architectural pattern used in your design (client-server, model-view-controller, etc.), and provide a high-level diagram (this diagram can be more informal and abstract than using any specific UML syntax). Explain why this architecture contributes to a good solution to the problem your system is supposed to solve.

Design decision rationale:
Identify a significant design decision for which you seriously considered one or more **alternatives**. Briefly describe each alternative, and discuss its pros and cons. Finally, explain why you finally chose the design you did.

Data organization description:
Describe, in detail, what **data** your system stores, and how. (A table that doubles as a glossary or data dictionary would not go amiss here.)

If your system uses a database, give the high-level database schema. If not, describe how you are storing the data and its organization (plain text, XML, graph, single file, multiple files, directory structure, etc.).

UML class diagram of detailed design:
Provide a UML class diagram of the detailed design of your system. Check the Links tab on the course website for some tools for drawing UML diagrams. It may be worthwhile to draw out your diagram on paper or a whiteboard first since you may go through several revisions.

When creating your class diagram, take care not to forget important classes that would reasonably needed to solve the task you are working on. Do not forget to include classes for **all aspects** of your system, such as user interface, data modeling, database interaction, any "helper" classes or utility code, etc. In the past, some groups have focused too much on user interface classes and have done a poor job capturing the details of the "model" of the system, the classes that actually represent the important data and behavior necessary for the task. (Some of these may be objects that come into being when needed and are destroyed when their usefulness is ended.)

Submission and Grading:

In evaluating your work, we will be looking to see that you have addressed all the necessary elements of a software architecture and made reasonable decisions related to all project components.

The UML class diagram must include enough detail to both answer questions about the design, and let the design be critiqued. In particular, your diagram should display all major classes, attributes (fields), methods (for brevity, do not list *get*, *set*, *is* methods), method visibility, inheritance/interface relationships, and associational relationships (named and directed, with multiplicity adornments).

Remember that part of your grade comes from having at least one meaningful in-person interaction with course staff before the end of Phase 2 to show your progress, ask questions, get feedback, etc., and generally make sure you are on the right track. (Phase 2 includes the deliverables Zero-Feature-Release and Design.)

Design Tips:

Your design should use design principles discussed in this class and prior ones. Here are some general design tips, which essentially come down to "put functionality in a logical place".

- Use encapsulation.
- Keep related data and behavior in the same place; emphasize **cohesion**, limit **coupling**, and minimize each class's public **interface**. (See slides in lecture #7, Architecture.)
- Avoid "god classes" that do too much; avoid insignificant or irrelevant classes that do too little.
- Every class should be able to be described concisely and clearly.
- Keep the model independent from the view.
- Allow for features to be developed in parallel as much as possible.