CS4911 Design Documentation

Architecture and Rationale

After you have completed an initial version of your Vision statement and use cases, you will have a list of customer features expressed as user stories in your Product Backlog. You next need to convert this into a design. The first step for your design is to create the Software Architecture for your application. There are many different architectures that you might develop for your system, and you will need to generate a few and then choose between them. You should document the reasons why you choose a particular architecture in your Design Rationale. Rationale is important for the long-term maintenance and enhance of your application. Unlike other programming projects you may have done in other classes, your Senior Design project will live on after you leave the class. This means your design information will be critical to the ability of your customer to maintain and evolve your application over time.

Be able to discuss in rationale the alternative designs and why you choose the one you did. If you only could come up with one design, you either have too simple a project, or haven't really considered all the options available. The *why* is typically expressed as enumerating the advantages and disadvantages of each approach.

Architecture is the highest, most abstract level of design. It should give a representation of how your application will interact with external entities and how it is organized. We talk about architecture all the time with standard terms like: Client-Server, Layered, Event-Driven, or Pipe and Filter. These are commonly called Architectural Styles. Your application may use one or more of these to help you achieve an optimal design for your system. Be sure to break down components as necessary. For instance a thin client might have just two boxes at the top (Client and Server), but the Server may actually have a complex architecture of its own.

It is important to provide both a static and dynamic view of your architecture. A static view might, for example, be expressed with a UML Package diagram. A dynamic view could be a system-level sequence diagram or a textual description of how functionality is realized at the architectural level. Each component should be indentified as to what

functionality it provides and how it interoperates with other components to form a working system. The dynamic description can be a textual walkthrough of the architecture along with a scenario, or it can be a sequence diagram.

Data

Many systems involve the management of persistent data, either through files or a database. How the data is organized is another aspect of design, and its expression is called a *data model*. If you have taken a database course, then you are already familiar with Entity-Relationship (ER) diagrams. These contain virtually the same concepts as do UML class diagrams, and the latter can be used if you are unfamiliar with the former. Your design documentation should include one of these visual descriptions of how any persistent data will be stored.

If your application uses flat files or XML data files, then the format for those files should be presented and documented.

Detailed Design

Decompose the high-level architecture into the lower level classes and document their dynamic behavior. If a non-OO application, document the static and dynamic behavior of the application. Typically, for UML this would be a class diagram and any of: state diagram, sequence diagram, collaboration diagram, or activity diagram.

For web applications, this might include a page navigation graph, along with summaries of the scripts running on each page.

GUI

Many modern systems are interactive. That is, the user interacts with the system using some form of display screen using which data can be entered and results presented. Well-designed GUIs are essential to successful systems because they are the most visible part of the system. As such, it is important to get early and frequent feedback from users about their reactions. This can take the form of a prototype, but if this is infeasible, then mocked up screen shots can be substituted. Your design documentation should include a description of your user interface design. You should explain what the viewer is looking at for each screen. Even command line apps have a UI and you should describe the command line options. If you are making a library rather than an application, then explain your API.

Validation

Every stage of software development should be validated. This is particularly true of design, because design is the most accessible description of the problem's solution. There are a variety of ways to validate software designs, but all of them involve comparing the design with the user needs. Examples include requirements tracing, non-functional requirements walkthroughs, or scenario-based evaluation methods. You should document the steps that you will take to validate your design. You may include the results here or in your individual design assessment.