CMPT370 – Design Doc outline and priorities

Things to keep in mind:

- A design doc is a way for us to communicate and justify our design decisions
    - Should be a clear explanation of our intentions
- In the real world we would have 2 design documents:
        1) For peer developers
        2) For the client
    - Since our Requirements doc wasn't so hot, we should perhaps consider making two via documentation reuse
- Let's meet with Dutchyn this week and clarify that we're on the right track
- All aspect of design must be justified. If a clear justification cannot be made, cut it.
    - These justifications must be clearly explained
    - If a design decision gives rise to new/unexpected benefits, document them
    - If a design decision gives rise to new assumptions, document them
    - If a design decision gives rise to new risks, document them
    - Let's sort out which aspects we can't justify clearly and discuss them with Dutchyn
- A good design doc outlines a common vocabulary which can be used between developers and clients to discuss the project
- A design document should provide a very clear logical understanding of the system before any implementation is attempted. This includes a vision for the organization of all objects, entities, and their relationships

Design Doc for a Peer Developer:

Section 1 – state the projects purpose:

- Begin with these concepts:
    - State the purpose of your project/system
    - What problem is it trying to solve?
    - Why does it need to exist?
    - Who will use it?
- Answering the above questions will help develop the scope of our design
- This section should be a few paragraphs long
    - If it's shorter, we might not understand the project well enough
    - If it's longer, the scope might be too large

Section 2 - Define the high level entities of the design:

- High level entities are objects or groups of objects that constitute major constructs of our design
    - Examples: the controller object, the data access layer, system security, etc.
    - In this section, explain in a few sentences what each entity does and justify why it is necessary to the system

Section 3 – Define low level design for each entity:

- The purpose of this section is to define objects and object relationships
- Usage:
    - Describe, in one paragraph per object, how each object is used and what function it serves
    - If an object will interface with an external object or system, show the interface for that object and the interaction
    - Re-describe the reason for including each object and again provide any benefits or risk factors involved with the object
    - If the object provides any encapsulation, briefly describe the encapsulation and why it is valuable
    - Use descriptions to give meaning to any diagrams, not the other way around
- Configuration:
    - If an object needs any special configuration or initialization, describe it in this section
    - If no configuration or initialization is necessary, leave this part out
- Model:
    - Define design model (i.e. MVC) and provide ample benefits and risks to this decision
        - Provide diagram(s) to supplement this description
    - Show the entire design with detailed UML diagrams
        - Provide descriptions in English
    - A descriptive explanation is more important than a perfect representation of the system in the models/diagrams
- Interaction:
    - Interaction diagrams for each relationship to show how a set of objects or entities communicate with each other to perform a complex task
    - Perfect UML is not necessary, focus on a clear description of a sequence of communications which accomplish a complex task
    - This section is to provide a high-level view of the systems interactions in order to examine them and ensure that they will function correctly

Section 4 – Benefits, assumptions, risks/issues:

- Make a list of the top 5-6 benefits of the design
- List all known risks
- List all known assumptions
- It is okay to reiterate points made in previous sections of this document, because the purpose of this section is to outline all benefits, assumptions, and risks in one area, so that a reader needn't read through the entire document to understand these concepts
- Never remove anything from this section
    - As risks become non-risks, document this change and the reason for it
    - As assumptions become non-assumptions, document this change and the reason for it
    - As benefits become none-benefits, document this change and the reason or it

Design Doc for a Manager:

- Made up of sections 1, 2, and 4 from the Peer Developer Design Doc

Dutchyn's Expectations:

The document is simple: three parts. First, describe and justify the architecture for your system. There are several architectures that will work...and exceptional students will provide a blend to highlight the communications/interactions in the system. Second, give a detailed design including UML class diagrams for all key classes that your system will contain... this should be detailed and complete enough that I can swap your design with another group and they will be able to construct the software. Third, enumerate and explain the differences that your designed system has from your required system ... what changed from the last stage?

To avert the typical questions:

How long should it be? Long enough to achieve the quality (grading standard) that you want and have resources (time) for: exceptional (90+)? excellent (80+)? good (70+)? satisfactory (60+)? minimally acceptable (50+)? See previous topics for other perspectives on this.

Do I have to give UML class diagrams for all classes? Of course not: only an exceptional document would contain that level of detail ... you decide what your grade will be by deciding the quality of product you produce.

How detailed should the justification be? A minimally acceptable justification (50%) needs to be more than "it works." An exceptional one might describe more than one possible architectural combination and provide an insightful and compelling comparison to satisfy the reader that the chosen architecture is strong.

Can we change some of the choices (e.g. supported number of players) from the requirements document we submitted? Of course, but you need to tell us what those changes are, and explain why you need to change them. Explanations like "I don't know how to do this" or "we're running out of time" are not ones that most customers would accept, and neither will we. Ultimately, you cannot remove the requirement that the system support 2, 3, and 6 players, both humans and scripts (using our Forth language encoded as JSON files), on the hex-grid.