About project deliverable 2: Design and Implementation documentation

Posted on: Tuesday, February 25, 2025 5:33:45 PM HKT

Dear all,

Here are the FAQ regarding design and implementation documentation:

1. **Any sample?** A sample can be found in Blackboard under "Course Contents" > "Project spec and materials" > Design & Implementation documentation sample: biking2

2. **What should be included?** The SRS document is a document for all stakeholders, including both non-technical end-users and technical developers. It lists the requirements that the software must fulfil. On the oth hand, the Design and Implementation documentation is a document for only technical developers to follow. The content should include everything that allow the technical developers to implement a software that all requirements from scratch, such as the design direction and rationale, how the responsibilities (functions to achieve) are divided among various places in the source code, the internal interfaces between compo the internal interfaces between functions and classes (if using OOP), and the external interfaces if the software is going to interact with other components to form a bigger system.

3. **How detailed should it be? Does that mean, e.g. the sequence diagrams for all functions in the software are required?** If the doc is too detailed, we may not want to read it if we are the developers 😀. It wo also be hard to maintain because the actual low-level implementation may change over time. We need to judge what should be detailedly described or in a relatively abstract manner. Recommended rule of thumb
    1. Include the design considerations for every requirement listed in SRS
    2. Consider describing the design in the least detailed level first. Then, imagine whether the readers would go wrong by reading only that level of description. If there is a chance that the developer will go wrong describe the design with more details.
        - For example, suppose we are implementing a database, and we need to find the largest number in the array to build indexes. We could write "1. sort the data in ascending order, 2. pick the last elemer describe the direction of implementing this function. The developer may have one question: which sorting algorithm should I use. If it does not matter, we do not need to specify which sorting algorithr use in the doc. Suppose we really need to specify the sorting algorithm as quicksort, then do we need to specify how quicksort is to be implemented (iterative vs recursive function calls)? We may not n do so if any kind of implementation can do; if the way of implementation does matter, we need to explain why we made the decision. Anyway, in the source code, the implementation is expected to ha documentation too.
        - Another example about an OOP design. Suppose we expect 10 classes are needed in the implementation. Only 3 of them are core classes whose designs are very important. Their attributes and intera must be understood properly. The remaining 7 are "helper classes" for the internal implementations of the 3 core classes' functions. How the 7 helper classes interact with each other does not matter design. They can even be replaced with another form of implementations. Then we may omit the details or even omit mentioning those 7 classes in the Design and Implementation documentation.
    3. In short: Just detailed enough for technical developers to follow to implement the system correctly and fulfil all the requirements. Some fine details can and should be put in the source code documentation
4. **I see "list of functions" for developers to reference in some open-source software. Do we need to include that?** No need, but it can be included in the appendix. It is known as the "API documentation" where available function interfaces of your library are listed. It can usually be generated automatically from the source code.

5. **What should really be included?** Recommended:
    1. Introduction - Basically a very brief summary of the requirements listed in the SRS
    2. System architecture (higher level)- Major components in the software system, what they are, how they relate to each other
    3. Data models (both high and low level)- Database schemas, flow charts, data flow: from end-user input, to final output, how is the data transformed?
    4. Interface design (higher level)- How internal as well as external components will communicate with each other, expected function inputs and outputs, communication protocols, expected exceptions and how are handled, security
    5. Component design (low level) - For every major component, specify its responsibilities, input and output specifications, algorithms, class definitions if OOP
    6. User interface design - Details about the user interface's layouts, "site map": what will be the next interface if the user enters something in interface X? Considerations about letting all users, including those w disabilities, to use your software. Can use simple drawings of user interface, known as the "storyboard", to show the flow
    7. Assumptions --- Hardware or software or other constraints such as "only one user at a time", assumptions about the development process e.g. need to use prebuilt virtual machines to standardize the development environment
    8. Adding some UML diagrams {component, class (if OOP), sequence diagrams in particular} can really improve readability, though not required

6. **Since this is a documentation of Design and Implementation, shall we include the implementation details, and use actual code instead of pseudocode for the design?** Yes. Just do not duplicate what's alr available in the source code documentation.

7. **Should we start coding now?** Yes, unless you adopt the "waterfall" development process. But it may be too late if you do not start coding now

8. **Grading criteria**, whether all requirements in SRS are considered, whether the design decisions are reasonable

Thanks for reading this looooong message.