<Project Name>
Software Design Specification [Template]

A. Hornof (ajh) - 5-6-2019 - v0.91

Table of Contents

(Read and then remove all of this text that appears before the start of Section 1.)

This document describes the content and general organization to be used for a software design specification (SDS) in Prof. Anthony Hornof's CIS 422 Software Methodologies class. The document is derived from several primary sources, each of which is cited and listed in the references at the end of the document. Most SDS templates, such as those provided by the IEEE (Institute of Electrical and Electronics Engineers), are longer and more complex than what is shown here. This template reduces the SDS down to a set of essential components.

How to use this document: Keep the headings and document structure, remove all of the text that describes what should go in each section, and replace it with the content described in that section. (To assist with this, this document will be made available in a few different file formats, such as PDF, Pages, and Word.) The final SDS should be a self-contained document that clearly describes what the document is, who wrote it, who wrote various sections, and when they wrote the sections. None of this explanatory text that describes *how to write* the document should be present.

1. SDS Revision History

This should list every modification to the document. Entries should be ordered chronologically, either in forward or backward chronological order; pick an order and stick to it. The entries listed in the template are for the creation of the template.

Date	**Author**	**Description**

4-30-2019	ajh	Created the initial document.
5-3-2019	ajh	Finished "Section 0 - How to use this template." with a discussion of IEEE concerns, viewpoints, and views.
5-6-2019	ajh	Removed the section entitled "If this template does not seem right for you…" which discussed the IEEE "concerns, views, and viewpoints".

2. System Overview

The system overview is a brief description of the services provided by the software and how the system is organized, with an emphasis on the software components and how they interact with each other. The overview can discuss but should not emphasize the user interface.

3. Software Architecture

(The term *module* is used here to describe a separable piece of software that is a part of the entire computer system, such as a "student records" module. The term *component* is used somewhat interchangeably.)

Software Architecture

The description of the software architecture should capture and communicate the important design decisions regarding how the system is decomposed into parts, and the relationships among those parts (Faulk, 2017). The architecture should describe:

1. The set of components. This should be both in easy-to-read list form, and also in a diagram.

2. The functionality provided by (or assigned to) each component. This should be included in the list of components.

3. Which modules interact with which other modules. Describe how the components work together to achieve the overall system functionality. This should be indicated in the architectural diagram, and also described in brief paragraph form after the list of components.

    This should be at a level of abstraction that you would use to explain to a colleague how the system works. It should be abstract and static, such as "The client database holds all of the client information and interacts with the data-cleaning module to ensure that no sensitive data gets released through the online system...." It should not be a detailed textual description of a dynamic flow of control such as "module A passes the record to module B which removes the user ID and then passes the record to module C...."

4. The rationale for the architectural design. This should be in paragraph or list form. Explain how and why this architecture was decided to be the best to solve the problem. See “Design Rationale” in the next section.

Do not keep your architecture simple just to reduce the number of modules you need to describe later in the SDS. If your architecture has only one module, your project may be too small for the purposes of an SDS, or too small for a class project, or there may be a problem with your design.

Use descriptive names for components.

Every module should have a name that is specific to the project. Do not use generic names such as "User Interface", "Model", "View", "Controller", "Database", "Back end", or "Front end". Instead, use names specific to the functionality of this system such as "Instructor Interface", "Student Interface", "Roster", "Student Records", "Roster View", "Grade View", and so on. Every module name should in some way convey the module's role *in this project*, not the role in a generic software design.

Do not name modules "client" or "server". The roles of client and server are relative to a particular service. A component can be a server with respect to one service and client with respect to another. (Faulk, Young)

4. Software Modules

4.1. <Module Name> (Include one subsection for each module.)

Each module should be described with:

a. The module’s role and primary function.
b. The interface to other modules.
c. A static model.
d. A dynamic model.
e. A design rationale.

The module’s role and primary function.

Every module needs to be described first abstractly in terms of its function or role in the system, and then in more detail such as in terms of its data structures and functionality. When describing such details, lists and diagrams will probably provide be more readable and searchable (for the eyes) than paragraphs of text.

Interface Specification

A software interface specification describes precisely how one part of a program interacts with another. (Faulk, Young) This is not the user interface, but instead a description of services such as public methods in a class, or getters and setters. Describe the software interface that each module will make available to other software components, both internal and external. This will help to explain how components will interact with each other, what services each module will make available, how to access a module, and what needs to be implemented within a module.

Find the right abstraction for each interface specification. For example, describe the services that will be provided but not how they will be implemented within the module. An interface that reveals too much information is *over-specified* and limits freedom of implementation. (Faulk and Young, 2011)

A Static and Dynamic Model

Every software module should be described with a static or dynamic model, and probably both.

Each diagram should use a specific design *language*. There are enough languages that have been developed such that you will not likely need to devise your own. Most diagram languages emphasize a static or dynamic representation, and do not typically mix the two. For example, class diagrams are primarily static, though they hint at time-based dynamic activities in their method names. Sequence diagrams emphasize activities over time, but also list the static entities (such as the classes) that are interacting. But you do not typically see a dynamic activity indicated on an association between two classes in a class diagram.

Every diagram should be introduced with a caption that appears immediately below the diagram and should what is show in in that diagram. Each caption should starts with "Figure <x>." and be referenced in the body of the text as "Figure <x>." The caption should briefly describe what the diagram shows, such as "Figure 3. A sequence diagram showing the 'Feed a child' use-case."

Diagrams can be hand-drawn and scanned in. There are some advantages to hand-drawn diagrams such as they are in some ways easier to modify. But they should ideally be scanned-in rather than photographed by hand, in part to keep the file sizes down. Do not fill an SDS with large (>1MB each) high-resolution photos of hand-drawn diagrams.

Design rationale

There will be a reason that each module (and the architecture) is designed as it is. The "design rationale may take the form of commentary, made throughout the decision process and associated with collections of design elements. Design rationale may include, but is not limited to: design issues raised and addressed in response to design concerns; design options considered; trade-offs evaluated; decisions made; criteria used to guide design decisions; and arguments and justifications made to reach decisions." (IEEE Std 1016-2009) Your design should find a good separation of responsibility for each module. One design rationale required by the IEEE standard

(1016-2009) is “a description of why the element exists, ... to provide the rationale for the creation of the element.”

Alternative designs

Your SDS for this class should include alternate designs that were considered for the architecture, and for each system or subsystem. This should result from either (a) considering multiple alternative designs considered during the project or (b) your design evolving over the course of the project.

These alternative design ideas and diagrams can be placed in a separate section entitled “Alternative Designs” or “Earlier Designs”, or could be placed immediately after each current design, and clearly marked as an alternative or earlier design.

This would not be part of a typical SDS but is included here to reinforce the idea that design is a process.

If you do not consider alternatives, you are not doing design. If you are not recording the alternatives that you consider, you are not engaged in a good design practice.

Describe alternative architectural decisions that were considered. (If there were no alternatives, then no “design” work was done.)

5. Dynamic Models of Operational Scenarios (Use Cases)

Every major Use Case should be described with a dynamic model such as a UML sequence diagram.

6. References

This section lists the sources cited in the creation of this template document. An SRS should reference all of the sources that it draws from. If sufficient citations are provided “in line” (at the point of reference) in the document, this section may not be necessary.

Faulk, Stuart. (2011-2017). CIS 422 Document Template. Downloaded from https://uocis.assembla.com/spaces/cis-f17-template/wiki in 2018. It appears as if some of the material in this document was written by Michal Young.

IEEE Std 1016-2009. (2009). IEEE Standard for Information Technology—Systems Design—Software Design Descriptions. https://ieeexplore.ieee.org/document/5167255

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Commun. ACM, 15*(12), 1053-1058.

7. Acknowledgements

Acknowledge any sources used in your document and your project.

This template builds slightly on a similar document produced by Stuart Faulk in 2017, and heavily on the publications cited within the document, such as IEEE Std 1016-2009.