**<Project Name>**

**<Team Members>**

**Software Design Specification [Template]**

***How to use this document:*** Keep the headings and document structure, remove all the text that describes what should go in each section, and replace it with the content described in that section. The final SDS should be a self-contained document describing the system design. Specify who and when wrote each section. None of this explanatory text that describes *how to write* the document should be present.

# System Overview

The system overview briefly describes the services provided by the software and how the system is organized, emphasizing the software components and how they interact with each other. The overview can discuss but should not emphasize the user interface.

# Software Architecture

(The term *module* is used here to describe an independent 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 software architecture description 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 should be in easy-to-read list form and a diagram.

2. The list of components must include the functionality they provide.

3. How modules interact with each other. Describe how the components work together to achieve the overall system functionality. Indicate it in the architectural diagram and briefly describe it in a paragraph after the list of components.

Describe the modules 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 should be in paragraph or list form. Explain how and why you decided this architecture was the best solution. 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 to be characterized by an SDS, 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 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)

# Software Modules

This section lists and describes the modules or components included in the system. Describe each module in a separate section.

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

Each module's description must include:

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 abstractly in terms of its function or role in the system and then its data structures and functionality. When describing such details, lists and diagrams will probably provide a more readable and searchable (for the eyes) than paragraphs of text.

Throughout the document, lists and sublists of design specifications should be indented and numbered to make it easy to read and reference the specification details. Such as:

1. *Module Name*

1.1. *Design Specification Detail*

1.1.1 *Design Specification Sub-Detail*

Note how this permits reference to "SDS Item 1.1.1".

***Interface Specification***

A software interface specification describes precisely how one part of a program interacts with another. A software interface is not the user interface but a description of services such as public methods in a class or getters and setters. Describe the software interface each module will make available to other internal and external software components. This description will help explain how components will interact with each other, what services each module will make available, how to access them, and how to implement each.

Find the right abstraction for each interface specification. For example, describe the services that modules provide but not their implementation. An interface that reveals too much information is *over-specified* and limits implementation freedom. (Faulk and Young, 2011)

***A Static and Dynamic Model***

Describe software modules with static or dynamic models - probably both. Each diagram should use a specific design *language*. There are enough modeling languages; 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 and list the interacting entities (such as the classes). However, you do not typically see a dynamic activity indicated on an association between two classes in a class diagram.

Every figure must include a caption immediately below it. Each caption starts with "Figure <x>." and is referenced in the body of the text as "Figure <x>." The caption briefly describes the figure, such as "Figure 3. A sequence diagram showing the 'Feed a child' use-case."

Diagrams can be hand-drawn and scanned in. Hand-drawn diagrams have some advantages, i.e., they may be easier to modify. But they should ideally be scanned in rather than photographed by hand, partly 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 to design each module (and the architecture) 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" (See 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 you considered for the architecture and each system or subsystem. These designs result from either (a) addressing multiple alternative designs considered during the project or (b) your design evolving throughout the project.

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

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

Document alternative architectural decisions you made. (If there were no alternatives, there was no design work).

# Dynamic Models of Operational Scenarios (Use Cases)

Describe every major Use Case using a dynamic model such as a UML sequence diagram.

# References

An SRS should reference all of the sources that it draws from. This section may not be necessary if sufficient citations are provided "inline" (at the point of reference) in the document.

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.

Class Diagram. In *Wikipedia*, n.d. <https://en.wikipedia.org/wiki/Class_diagram>.

Sequence Diagram. In *Wikipedia*, n.d. <https://en.wikipedia.org/wiki/Sequence_diagram>.

# 7. Acknowledgments

List here all sources you used to create the document and support you received from anyone not on your team.