Software System Design Description

Version <number>

<date>

<Project Title>

<Author(s)>

# 

# Table of Contents

<generate here >

# List of Figures

# 1.0. Introduction

## 1.1. Purpose

< Clearly state the purpose of this *document* and its intended audience. >

## 1.2. Scope

< State the dominant design methodology. Overview the architecture of the product briefly. Describe the external systems with which this system must interface. >

## 1.3. Glossary

< Define the technical terms used in this document. Do not assume the experience or expertise of the reader. >

## 1.4. References

< List here any references to other documents cited anywhere in this document including references to related project documents, especially the SRS. Add references here when other project documents are created. This is usually the only Bibliography in the document. >

## 1.5. Overview of Document

< Describe the contents and organization of the rest of this document. Since there is already a Table of Contents, this overview will be less formal but more informative. Describe the remaining sections. >

# 2.0. Deployment Diagram

< The deployment diagram provides a physical look at the system with each processor and device indicated. This provides a background for the rest of the document as no software component can straddle two physical locations. Each physical location will have its own software unit and units in different physical locations will collaborate to provide the services that logically seem to be straddling the units. >

# 3.0. Architectural Design

< A software system is a set of communicating entities that collaborate to perform a task. The Architectural Design shows these entities, their relationships and the relationship to the actors in the system. This top level is a diagram where each entity has a name, a (proto-) type, an abstract specification and an interface design. The abstract specification is a description of its purpose, its functionality, its attributes (including dependency on other entities) and the constraints under which it must operate. It also describes resources, that is, any elements used by the entity which are external to the design such as physical devices (e.g., printers), software services (e.g., math libraries) and processing resources (e.g., buffers). The interface design is the list of the services that it provides to clients. These services are methods (procedures and functions), each carefully documented using a pre-condition/post-condition formalism.

Each entity in turn may provide its services by having an internal architectural design with its own set of subordinate entities. These entities may be called sub-systems, components, modules or classes. The decomposition of a higher-level entity into subordinate entities must be explicit. The algorithm that shows how each method of the larger entity is performed by these components must be explicit. Any data stored in an entity must be explicitly described (see Data Structure Design below).

Note that while the abstract specification (architecture) and the interface (detailed) design are usually developed separately in an iterative approach, they are combined for documentation. >

# 4.0. Data Structure Design

< If not already covered above, design in detail and specify the data structures to be used in the implementation. If these include databases, define the table structure of all databases including full field descriptions and all relations. Graphical languages are appropriate. Note that a database is an object and may have been fully described in the previous section. If the material is covered under Architectural Design, this section is omitted. >

# 5.0 Use Case Realizations

< For each use case in the Requirements Specification there must be a use case realization here. That is, there must be a sequence of events using the design objects that will perform all of the operations promised in the SRS. It is possible that there will be several use case realizations here to show all the alternative and exception paths required. The relationship between use case specifications (full descriptions in the SRS) and use case realizations must be bi-directional (referenced from SRS to here and from here to SRS) and explicit. Each use case realization must also be cross-referenced to the use case test in the test design. This relationship between use case specifications and use case test must be bi-directional and explicit. >

# 6.0. Real-Time Design

< If relevant, tell how your design that will ensure that the real-time timing constraints will be met. If not relevant, omit this section. >

# 6.0 User Interface Design

< Refer to User Interface material in the SRS and supplement with any design considerations not mentioned there. You should discuss the expected effectiveness of your design. >

# 7.0 Help System Design

< Describe the **structure** of the help system and how it is to be accessed. Will it be context sensitive? Will there be a system reference capability? >

# Index

< generate here >