<Company Name>

<Project Name>
Design Guidelines

Version <1.0>

[Note: The following template is provided for use with the Rational Unified Process. Text enclosed in square brackets and displayed in blue italics (style=InfoBlue) is included to provide guidance to the author and should be deleted before publishing the document. A paragraph entered following this style will automatically be set to normal (style=Body Text).]

[To customize automatic fields in Microsoft Word (which display a gray background when selected), select File>Properties and replace the Title, Subject and Company fields with the appropriate information for this document. After closing the dialog, automatic fields may be updated throughout the document by selecting Edit>Select All (or Ctrl-A) and pressing F9, or simply click on the field and press F9. This must be done separately for Headers and Footers. Alt-F9 will toggle between displaying the field names and the field contents. See Word help for more information on working with fields.]

Revision History

Date	Version	Description	Author
<dd/mmm/yy>	<x.x>	<details>	<name>

Table of Contents

Design Guidelines

1. Introduction

*[The introduction of the **Design Guidelines** provides an overview of the entire document. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and overview of this **Design Guidelines**.]*

1.1 Purpose

The purpose of this document is to communicate the design standards, conventions and idioms to be used in the design of the system.
*[Enter any additional description of the objectives of the **Design Guidelines**.]*

1.2 Scope

*[A brief description of what the **Design Guidelines** applies to; what is affected or influenced by this document.]*

1.3 Definitions, Acronyms, and Abbreviations

*[This subsection provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the **Design Guidelines**. This information may be provided by reference to the project's Glossary.]*

1.4 References

*[This subsection provides a complete list of all documents referenced elsewhere in the **Design Guidelines**. Identify each document by title, report number (if applicable), date, and publishing organization. Eventually the section may be structured in subsections: external documents versus internal documents or government documents versus non-government documents and so on. Specify the sources from which the references can be obtained. This information may be provided by reference to an appendix or to another document.]*

1.5 Overview

*[This subsection describes what the rest of the **Design Guidelines** contains and explains how the document is organized.]*

2. General Design and Implementation Guidelines

[This section describes the principles and strategies to be used while designing and implementing the system. In most cases, you will need strategies for the following:

- ***Mapping from Design to Implementation***

  You must specify how the design is mapped to the implementation; both at the package level and at the class level.

- ***Specifying Interfaces on Subsystems***

  When you are developing the system from the top down, it's important to narrow the visible interfaces to the subsystems. This enables developers to change the parts of a subsystem that are not visible outside.

- ***Documenting Operations***

  It is important that you decide on a standard way of describing operations. An operation consists of the name, the arguments, a brief description, and an implementation specification. Ask yourself the following questions when documenting operations.

  — *Will all (formal) arguments be documented? It is suggested they need to be, although experience*

shows that they might be difficult to maintain because they are redundant with the code.

- *Will the argument type be documented? Generally it is suggested that it is.*
- *Will you use any naming convention for the operations in a class? For example, in C++ you might choose to prefix private and public operations in different ways. This makes the operations easier to understand.*

- ***Documenting Messages***

  We suggest that you do not document all actual parameters in the message. It's redundant with the code and might prove difficult to maintain.

- ***Detecting, Handling, and Reporting Faults***

  You must have a strategy for fault management. Your strategy depends to a large degree on the programming language you have chosen. Many languages feature fault-management support, such as Ada "exceptions." The fault-management strategy you choose will influence behavior in the design objects. For example, you must decide whether to use status parameters in each operation that tell if the operation has succeeded or to let the object raise an "exception," as in Ada.

  If necessary, you can apply different fault-management strategies to different parts of a system. The important thing is that you have at least one strategy, and for all possible strategies, that it is clear when to use them.

- ***Memory Management***

  Memory management means ensuring that memory is always available. This implies that you remove objects not referenced by any other object so that the memory they occupy can be used for new objects. How you solve this depends on the implementation language. In some systems it will be automatically solved, for example, by a garbage collector; but in others you must carry out the memory management yourself in the programming language. In other words, you will have to define when and how to clear any memory occupied by un-referenced objects.

- ***Software Distribution***

  If you have a system that will be distributed among several physical nodes, its objects must also be distributed among the nodes. Before design starts, you will prepare this work by specifying general strategies for how objects will be distributed, and how to use the present inter-process communication technology. If the target environment is unfamiliar, it might prove useful to prototype solutions.

- ***How to Represent Reusable Components***

  Before starting design, you must decide what reusable components, reusable component systems, libraries or "Commercial-Off-the-Shelf" (COTS) products to use. You must also decide if, and how these will be modeled in design.

- ***Designing Persistent Classes***

  Ideally, the database-management system you choose, whether relational or object-based, will not affect the design model very much. Persistence is provided by a framework that makes persistence as transparent as possible.

  Most persistence design work focuses on identifying and resolving performance problems. To make this easier, the following should be done:

  - *Identify the lifecycle of each persistent object: when it will be created, read, updated and deleted within Use-Case Realizations.*
  - *Identify transaction boundaries within Use-Case Realizations.*

There is some iteration between Database Design and the design of persistent classes: depending on the database, some associations between design classes are either clumsy to support in the database, or create such a performance problem that some adaptation of the Design Model is necessary.

- ***Transaction Management***

  This section discusses the strategies used to manage transactions, including how transaction management will be accomplished. The interaction of transaction management and Fault Management is discussed, including how the system recovers from transaction failures or aborted transactions.

  If there are special restrictions imposed by the transaction management mechanism (such as MTS requiring 'stateless' objects) that affect the architecture of the system, they should be discussed here.

- ***Special Use of Language Features***

  This section describes any restrictions or policies on the language features used. For example, you may limit the use of pointers in an embedded real-time system.

- ***Program Structure***

  This section includes guidelines for code layout, comments, naming conventions, module packaging, and interface conventions.

- ***Algorithm Guidelines***

  This section describes particular algorithms selected for use in the system by the software architect. The section also describes the circumstances under which use of the algorithm is appropriate. This section does not document particular applications of an algorithm in the system—that is the province of the Software Architecture Document and the Design Model—rather it is intended to guide and constrain the designer's choice.

- ***Hardware Interfacing***

  This section describes guidelines for hardware interfacing, including the use of interrupts, memory, type representation, and so forth.

- ***System Modification and Build Guidelines***

  This section describes guidelines for software modification, special support hardware or software needed, edit, compile and integration guidelines, and so on. This section may also contain configuration and change control guidelines elaborated from the Configuration Management Plan.

- ***System Diagnostic Guidelines***

  This section describes guidelines for system setup to diagnose problems, based on the fault detection and management strategy adopted. This section describes any special diagnostic hardware or software that may be used, how to invoke traces and profilers, and how to collect diagnostic data.]

3. Database Design Guidelines

[This section gives rules and recommendations for the database design. The following topics should be discussed:

- *Mapping from persistence classes to database structures, including how to handle potential conflicts such as many-to-many associations in the design model and inheritance.*
- *Mapping of design class attributes to database primitive data types.*
- *Use of the Process View to describe the processes and inter-process communication used by the persistence mechanism.*
- *Use of the Deployment View to describe the physical distribution of data across nodes.*

- *Naming conventions for database structures; for example, tables, stored procedures, triggers, tablespaces, and so forth.]*

4. Architectural Design Guidelines

[This section gives rules and recommendations for software architecture design. They are organized around the different architectural views: Use Case, Logical, Implementation, Process, and Deployment Views. The rules deal mostly with decomposition. For example, the Implementation View guidelines specify the rules for packaging modules into subsystems, layering subsystems, and so on. See the Software Architecture Document*, the section titled* Analysis & Design *in particular.]*

5. Mechanism Guidelines

[For each significant mechanism put in place in the low-level layers of the system, you should have a programmers' guide that shows the interface of the mechanism and explains how to use it. These include a user's guide of the timer mechanism, of the inter-process communication mechanism, of the recording mechanism, of the database-management system, and so on.]

6. UML Stereotypes

[This section contains or reference specifications of Unified Modeling Language (UML) stereotypes and their semantic implications—a textual description of the meaning and significance of the stereotype and any limitations on its use—stereotypes already known or discovered to be useful for constructing Design models. The use of these stereotypes may be simply recommended or perhaps even made mandatory; for example, when their use is required by an imposed standard, when it is felt that their use makes models significantly easier to understand, or when it ensures that common types of entities, roles, relationships, or patterns are uniformly modeled and understood. This section may be empty if no additional stereotypes, other than those predefined by the UML and the Rational Unified Process, are considered necessary.]