Introduction

Documenting a design involves describing the structure and behavior of the software from a number of perspectives

The purpose of a design document is twofold:
1. To show how the software design meets all of the requirements.
2. To communicate an understanding of the application domain.

These instructions describe use of the object-oriented design template.

Architecture

Create and insert one or more diagrams that show the high-level design of the application.

The types of diagrams you may use to show your application architecture can include one or more of the following. Please note that these diagram types are not listed in order of preference.

- A UML class diagram that shows how classes are related to each other. Each class would have a name but would likely not show any attributes or methods.
- A UML package diagram that shows the major components of the design and how the classes in each package are related to classes in other packages.
- A data flow diagram (DFD) that shows the major components of the design and how these components are related to each other via data flows and data stores.
- An ICOM/IDEF function model that shows the major components of the design and how these components are related to each other via input, control, output, and mechanism data flows.

Data

This section contains a description of the logical and physical view of the data used by the application.

Logical Data Model

Create and insert a logical data model (LDM) for the application. This should show the strong entities, weak entities, and the relationships between each data entity. The model should show the attributes associated with each data entity and which attributes represent the primary key.

After the LDM, list the rules shown in the logical data model. Each rule is a single sentence that describes a relationship between two logical entities. A few example rules are listed below, based on a LDM that describes contact information. (The text in parentheses is an explanation of what the LDM would look like in order to generate this rule. This text is part of these instructions and not intended to be included in your design artirfact.)

- A contact has a name that is not required to be unique. (The LDM shows a contact entity that has an id attribute as the primary key and a name as a non-key attribute.)
- A contact has zero or more phone numbers. (The LDM shows a relationship between a contact entity and a phone number entity. The cardinality shows that a phone number instance is not required for each contact instance.)
- A contact helps to identify their phone numbers. (The LDM shows a relationship between a strong contact entity and a weak phone number entity.)

Physical Data Model

When a persistent data store is being used by the application, create and insert a physical data model to show the physical representation of the data as it will exist in the persistent data store. When a persistent data store is not being used, this section may be removed from your design artifact.

The type of physical data model used is based on the type of persistent data storage to be used by the application. The following is not intended to be an exhaustive list of the types of persistent data stores that may be used by an application.

- For a text file, this section needs to describe the contents and format of the lines of text. When there are different types of text lines in the text file, each type of text line needs to be described.
- For a text-based XML file, this section needs to describe the hierarchy of XML tags that describe the structure of the data.
- For a relational database, this section needs to describe the tables, primary keys, foreign keys, non-key fields, and indices based on the logical data model.

Internal Data Structures

When non-persistent data structures (e.g., linked list, array, tree, graph, document object model) will be used by the application, provide a description for each data structure. Otherwise, this section is not applicable and may be removed from your design artifact.

Data Dictionary

List the attributes shown in the logical data model and describe their physical characteristics. Use one of the following tables based on the type of physical data store being used.

All three tables shown below contain the following columns:

- Attribute the name of the attribute on the logical data model.
- Type the type of data stored in the attribute.
- Size the size of the attribute, based on its Type.
- Format/Range any special format rules or range of values the attribute must adhere to.

For a text file:

- Structure the internal data structure(s) that store values associated with this attribute.

Attribute	Type	Size	Format/Range	Structure

For a text-based XML file:

- XML Tag the XML structure(s) that store values associated with this attribute.

Attribute	Type	Size	Format/Range	XML Tag

For a relational database:

- Table the table(s) that store values associated with this attribute.

Attribute	Type	Size	Format/Range	Table

Interfaces

This section contains a description of two types of interfaces: the human-computer interaction and the ways that the application will communicate with the external entities as described in the Architecture section.

When one of these interface types does not exist for your application you may remove that section from your design artifact.

Human-computer Interaction

When the application has an interface with a human user, create and insert a state machine diagram (aka state chart) that explains how the user would navigate through the application. When the HCI design is large or complex, you may want to have more than one state machine diagram or you may want to include descriptive text that helps explain the interactions between the application and a user.

In addition, when the HCI is either large or complex, you may want to include one or more pictures to illustrate its appearance. When doing this, adding text to connect the state machine diagram to the picture would help in understanding the HCI design.

Communication with External Entities

When the application must communicate with one or more external entities (as shown in the Architecture section), create and insert a class diagram that emphasizes the way in which the application will communicate with each type of external entity. The class diagram would show the API (application programming interface) to be used between your application code and the external entity.

Components

This section contains a more detailed description of each component described in the Architecture section. Create a separate ***[Component X]*** section for each component and rename each section to one of the component names.

[Component X]

Each component in your architecture/design will have a separate section that shows a class diagram (i.e., a structural model) and a behavioral model of the component.

You will need to copy this section for each component that exists in your architecture/design. Change the [Component X] heading for each section to the name of the component being described.

Class Diagram

Create and insert a UML class diagram for component X. This class diagram should, at a minimum, show the public elements found within the component. These public elements include, but are not limited to, classes, abstract classes, interfaces, enumerations, constructors, methods, static attributes, and constants. When other non-private access modifiers (e.g., Java `protected`) are used in the internal design of the component, these design elements should also be shown on the class diagram.

Behavior

Some processing performed by the component may be unique or complex. For purposes of these instructions, *unique processing* refers to requirements that are not familiar to the designers/developers while *complex processing* refers to the involvement of many objects, methods, and attributes. When the processing is unique, complex, or both, create one or more behavioral diagrams that describe how the objects and methods are combined to implement the necessary behavior.

You may use one or more of the following UML behavioral diagrams to model the behavior of the component:

- Activity diagram – shows a workflow as a series of activities that may be performed sequentially, iteratively, by choice, or in parallel.
- Communication diagram – shows how objects communicate with each other via messages (i.e., method calls). This type of interaction diagram numbers the messages to show ordering of method calls.
- Sequence diagram – shows how objects communicate with each other via messages (i.e., method calls). This type of interaction diagram uses swim lanes and object life lines.
- State machine diagram – shows a collection of states and how transitions occur between these states.
- Timing diagram – similar to a sequence diagram; this shows messages that progress in time based on their relative position within the diagram. Messages to the left occur before messages to the right.

Communication between Components

When your architecture/design contains two or more components, this section should describe how these components communicate with each other.

Class Diagram

Create and insert a UML class diagram that shows the classes that are responsible for communication between the components of your design. This class diagram should, at a minimum, show the public elements found within each class that allows one component to communicate with one or more other components. These public elements include, but are not limited to, classes, abstract classes, interfaces, enumerations, constructors, methods, static attributes, and constants. When other non-private access modifiers (e.g., Java `protected`) are used in the internal design of these classes, these design elements should also be shown on the class diagram.