CMSC 345

Software Design and Development

(Adapted from Susan Mitchell)

System Design Document Template

Writing Instructions

Use the materials posted under the Writing Resources button on Blackboard as references for grammar, spelling, punctuation, formatting, and writing style.

Be sure that your document is

- Complete - No information is missing
- Clear - Every sentence's meaning must be clear to all parties
- Consistent – The writing style and notation is consistent throughout the document and the document does not contradict itself
- Verifiable - All requirements and other facts stated are verifiable

Remember that you are required to do a team review of this document.

When you think you are done with the SDD, ask yourself, "Could someone who was not part of the development of this SDD write the corresponding code?"

[Put product name here]
System Design Document

Table of Contents

1. **Introduction**

1.1 Purpose of This Document

State the purpose of the document and specify the intended readership. Briefly summarize the content. [One paragraph]

1.2 References

Provide a list of all applicable and referenced documents and other media. Minimally, references to the SRS and UI Design Document go here. If you used any other documents or references to arrive at this design (e.g., the Somerville text, UML references, documents provided by the customer, websites), list them here. See the Writing Resources on Blackboard for the appropriate formats for references.

2. **System Architecture**

2.1 Architectural Design

Draw a diagram that represents the logical architecture of your system. Describe what you are trying to communicate to the reader with the diagram.

Describe the technology (hardware and software) that your system will use. If your system is more than a desktop system (e.g., a client-server system), draw a corresponding technology architecture diagram.

Illustrate and describe as much as you currently know about the system's architecture. Use the architecture drawings in Mr. Roache's *Software Design: Requirements to Code* lecture as a guide. You may also want to search the web for other examples.

[Two to three substantial paragraphs]

2.2 Decomposition Description

Illustrate and describe the decomposition into components of the system that you presented in Section 2.1 (e.g., functions, objects, scripts, files). This is the view of the system as you know it at this point in time. Provide diagrams as follows:

Functions – If your system is purely procedural, illustrate the interrelationships of its functions using a structural decomposition (hierarchy) diagram. Label the connections (interfaces) with general names (e.g., "student information").

Object-oriented – If your system is object-oriented, use class diagrams to illustrate the implementation design of your system. Include any attributes and methods (public, private, and protected, constructors, accessors, and mutators) that are known as of this time. Use the UML class diagrams reference at the end of this document as a reference.

Hybrid – If your system contains both functions and classes, you need to include both a structural decomposition diagram and implementation class diagrams.

Other – If you are using PHP or another language that cannot be adequately described using a structural decomposition diagram and/or class diagrams, create custom illustrations. Show implementation components of the system (e.g., scripts, files) and their relationships.

Regardless of the type of diagram(s) that you use, refer the reader to the diagram(s) and describe what it is intended to communicate. Give a brief description of each of the components. If you are using a pre-defined pattern (e.g., Model-View-Controller), explain this to the reader. [Two to three substantial paragraphs]

3. **Persistent Data Design**

3.1 Database Descriptions

Describe the database(s) used by the system. Include a diagram of the schema. You may use an entity-relationship (ER) diagram, or you may create your own diagram type (e.g., a collection of tables). For each field in a database, give its name, data type (e.g., int, double), size (e.g., strings), and description of what it represents. Basically, give all of the information that a programmer must have to implement the database. If no databases are used, simply state so. [Length is whatever it takes]

3.2 File Descriptions

Describe the file(s) used by the system. Include a diagram of the file structure. For each field in a file, give its name, data type (e.g., int, double), size (e.g., strings), and description of what it represents. Basically, give all of the information that a programmer must have to implement the file. If no files are used, simply state so. Supplement your description with a sample file(s). [Length is whatever it takes]

4. **Requirements Matrix**

Use a tabular format to show which system components (e.g., functions and/or methods) satisfy each of the functional requirements from the SRS. Refer to the functional requirements by use case number and name.

Appendix A – Agreement Between Customer and Contractor

Place on a separate page. Describe what the customer and your team are agreeing to when all sign off on this document. [One paragraph] Include a statement that explains the procedure to be used in case there are future changes to the document. [One paragraph] Provide lines for typed names, signatures, and dates for each team member and the customer. Provide space for customer comments.

Appendix B – Team Review Sign-off

Place on a separate page. Provide a brief paragraph stating that all members of the team have reviewed the document and agree on its content and format. Provide lines for typed names, signatures, dates, and comments for each team member. The comment areas are to be used to state any minor points regarding the document that members may not agree with. Note that there cannot be any major points of contention.

Appendix C – Document Contributions

Identify how each member contributed to the creation of this document. Include what sections each member worked on and an estimate of the percentage of work they contributed. Remember that each team member <u>must</u> contribute to the writing (includes diagrams) for each document produced.

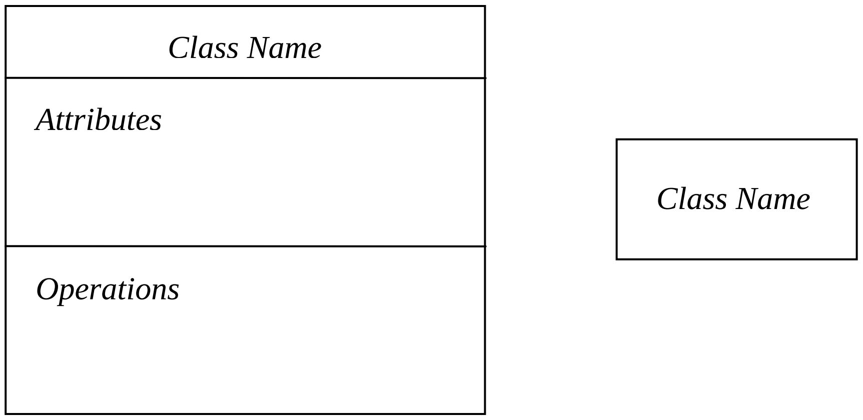
Unified Modeling Language (UML)
Class Diagrams

Reference: **UML Distilled**, 2nd edition, Martin Fowler and Kendall Scott, 2000

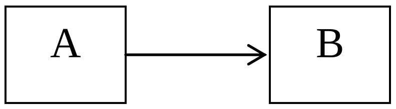
Note: The information below has been modified slightly to meet the purposes of CMSC 345

<u>Symbols:</u>

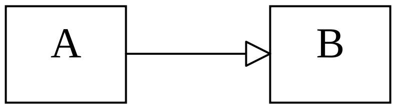
Class – Represented by a box as follows:



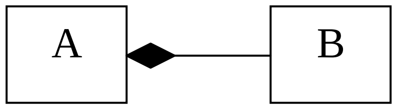
Navigability – Represented by a solid line with an arrowhead at one or both ends (unidirectional or bidirectional association, respectively). Indicates the direction(s) of an association between classes. (Below, A can navigate to B, but not the reverse.)



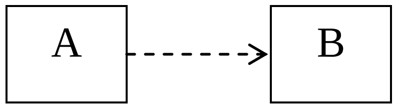
Generalization – Represented by a solid line with a hollow arrowhead at one end. Indicates that one class is a generalization of another. (Below, B is a generalization of A.)



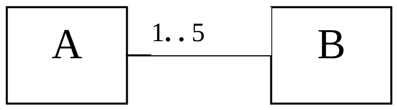
Composition – Represented by a solid line with a solid diamond at one end. Indicates that an instance of one class is “owned” by a single instance of another. (Below, an instance of B is owned by a single instance of class A. Note: if an instance of A is deallocated, the associated instance(s) of B are also deallocated.)



Dependency – Represented by a dashed line with an arrowhead at one end. Indicates that one class depends on the interface to another class. (Below, an instance of A is dependent on the interface to an instance of B.)



Multiplicity – Indicates how many instances of one class type are associated with another class. (Below, 1 to 5 instances of class A are associated with class B.)



Note: We will not be using any other symbols in our class diagrams.

Format for class attributes:

visibility name : *type* = *defaultValue*

where *visibility* = + for public, # for protected, - for private
name = attribute name
type = data type
defaultValue = default value

Format for class operations:

v*isibility name* (*parameter-list*) : *return-type*

where *visibility* = + for public, # for protected, - for private
name = operation name
parameter-list = comma-separated parameters with the syntax

direction name : *type* = *defaultValue*

where *direction* = in for input
out for output
inout for input/output
name = parameter name
type = parameter type
defaultValue = default value
return-type = return type