**Section 1 – State the purpose of your project/sub-system:** In this section, write a few paragraphs that describe what the project or sub-system does. What is the problem it is trying to solve? Why does it need to exist? Who will use it? By answering these questions, you establish the scope of your design. If you find it hard to write a few paragraphs in this section, then you probably don’t understand the domain as much as you should. If you can’t fit your description within a few paragraphs, then perhaps the scope is too large. Use this section as a tool to verify that the scope of your design is reasonable.

**Section 2 – Define the high level entities in your design:**

**Application:**

In the progression of the final product, the functionality begins with an application. The application effectively takes user input and translates that into something that the database language can understand. It is the intermediate step between the user and the database and provides a layer of abstraction which allows only desired access to the database information itself.

**Parser:**

The next layer is the parser. The parser acts as part of the database which converts human-readable language (ie. INSERT…, JOIN…, etc.) and turns it into the commands and functions which the database takes. It provides the arguments to these functions as well. It is necessary to define this step as it undoes the abstraction created in the application, which is necessary for *portability* of the final product; that is, the parser is the layer that allows any program written in the defined language to be run on our database. Again, note in the Figure 1 how many applications are able to run on the parser. Of course, in this project we only look to create one main application, but we retain the option of general use.

**Database:**

The next layer is the database, which is essentially a collection of relations. The database needs to maintain fluid control over all of its relations, and be able to add, remove, and access relations with ease. After the parser turns the language into functions and methods, the database needs to decide which relations it is looking for, determine if those relations are valid, and act upon the command it is passed.

**Relation:**

The final entity is the relation itself. A relation is a table, which contains columns called attributes (ie. *Name, Age, etc.*) and rows which contain values that make up entries (ie. *[John Smith, 19], [Bob Joe, 18]*). The relation needs to have quick access to all of its entries, and be able to return information about the entries it contains.

**Section 3 – For each entity, define the low level design:** This section is where your objects and object relationships are defined. For each object (or set of objects) define the following:

**Usage**

Describe in a paragraph how the object is used and what function it serves. If an object will interface with an external object or system, it is a good idea to show the interface for the object. Most importantly, you must again describe your thought process for defining the object as you did. List the benefits and risks. If an object provides an encapsulation, describe in a sentence why the encapsulation adds value. Use your descriptions to give meaning to the diagrams. They don’t have to be verbose, just enough to get the point across.

**Configuration**

If your object needs any special configuration or initialization, this is a good place to describe it. If not, this section can be left out.

**Model**

Figure 2 shows an example of a to supplement the System Security entity from figure 1. It is not perfect UML, but has some aspects of UML. Most importantly, it describes the design.

[](http://blog.slickedit.com/wp-content/uploads/2007/04/figure2.jpg)

Figure 2 (click to see full size)

Don’t worry about perfection in your models, but be sure to describe exactly what is going on in the diagram. Here, two concrete security objects derive from a base security object, and a security factory will create one or the other for a client depending on the security model of the system.

**Interaction**

This is also a good section for interaction diagrams. An interaction diagram shows how a set of objects or entities communicate with each other to perform a complex task. Figure 3 shows an example of an to show how a user might log in. It uses objects from the various entities shown in figure 1.

[](http://blog.slickedit.com/wp-content/uploads/2007/04/figure3.jpg)

Figure 3 (click to see full size)

Again, this diagram is not perfect UML, but it explains the communication sequence to accomplish a complex task. Interaction diagrams are most useful when you want to diagram how an object in your system will communicate with an object in another subsystem. This type of diagram will let the other developer verify that the interaction is correct.

**Section 4 – Benefits, assumptions, risks/issues:** In this section, make a list of 5-6 top benefits of the design, a list of **ALL** known risks/issues and a list of ALL assumptions. Some of this may simply be rehashing what you wrote in a previous section of the document. What’s important is getting all of these items into one section so that the reader doesn’t have to read the whole document to understand what the benefits, risks and assumptions are.

Never remove anything from this section! As risks become non-risks, document that they are now non-risks and why they became non-risks. Never erase them from the document. The same holds true for assumptions. You should be able to look at this section and know instantly what the current risks are to your design.

-The way we are storing the relations in files is a risk. Therefore, we assume that the program will not be overloaded.