**Moving into Program Design: A Review of DFDs**

**Content Author:** Terri Washburn, Ph.D.

We are methodically moving closer to the point where we are concerned with '**how'** the elements of the new system (or the project) will **interact** with one another. This week we will discuss **program design**, which takes us a step closer to informing the programmers / developers how to construct the coding logic to carry out our new system. Up until now we have been talking with stakeholders, examining the existing system, carefully gathering requirements, putting them into easily-understood diagrams and process flows, and figuring out what we are trying to accomplish without too much thought about 'how' it will do so.

To move forward, let's go back a few weeks first. Remember back in the second lecture in week 3, we introduced the concept of **data flow diagrams (DFDs)**, which were an extension of **use cases**? Remember that each of these diagrams focused on a different **type** of representation among elements in the system.

Test yourself for a moment on the purpose behind use cases and data flow diagrams:

|  |  |  |
| --- | --- | --- |
| Use cases told us... what?  [Click for answer](javascript:setContent('ans1',%20'link1');) |  |  |

|  |  |  |
| --- | --- | --- |
| Data flow diagrams are created from use cases and indicate...what?  [Click for answer](javascript:setContent('ans2',%20'link2');) |  |  |

|  |  |
| --- | --- |
| Here is a simple **DFD** to jog your memory: | Diagram  Description automatically generated |

Remember that there are specific **symbols** for the data flow diagram, as seen below from lecture 2 in week 3.

|  |  |
| --- | --- |
| **Elements of a Data Flow Diagram** | **Symbols** |
| * A business process (an activity) or processes | A picture containing mirror  Description automatically generated |
| * Data flows (sometimes called a message) | Shape, rectangle  Description automatically generated |
| * Data store (i.e., a file or database) | A picture containing text, table, dining table  Description automatically generated |
| * External entity or entities (a person, a system, or an organization that is external to the process). | Shape, square  Description automatically generated |

Now, we revisited all of this because program design asks that we move from logical data flow diagrams (which are what we've done, above) into physical data flow diagrams. What is the difference?

Here is one of the best clarifications between the two I've found, and I couldn't put it any better myself:

|  |  |  |
| --- | --- | --- |
| **Logical data flow diagrams** |  | **Physical data flow diagrams** |
| * Implementation-independent   and describe the system, rather than how activities are accomplished. |  | * Implementation-dependent   and describe the actual entities (devices, department, people, etc.) involved in the current system. |

*Retrieved on August 22, 2014 from site:*[*http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc38088.1610/doc/html/rad1232026266129.html*](http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc38088.1610/doc/html/rad1232026266129.html)

In other words, we move from "student fills out order form which gets stored in the order file" (which is high-level and somewhat vague) to a more **system-specific** and **organization-specific** diagram.

Move on to the next lecture where we will work from a logical DFD to a physical DFD

# Reality Check!

**Content Author:** Terri Washburn, Ph.D.

Diagram

Description automatically generated

It's time for a reality check here. All the diagrams in your textbook and in my lectures are professionally rendered, neat and easy to read. The reality is quite often analysts, database administrators and programmers spend a lot of time **sketching** out what they want to see. It's much quicker and a better use of "brain time" than dragging and dropping diagrams onto a drawing pad on your PC. Before you work to align all the boxes and arrows it can help to sit with a pad of paper and pencil and sketch out what your DFD should look like. Get those thoughts down quickly!

Diagram

Description automatically generated

Don't get me wrong: a formal DFD (or set of DFDs) is needed for documentation. Many organizations require them as part of the project documentation process or even as input to the data modeling that will follow. But there is nothing wrong with a paper rendition before opening up the application to create the real thing. After all, sometimes at the beginning, the thought of completing your diagram is not unlike this cartoon below:

Diagram, schematic

Description automatically generated

So go ahead and sketch out your thoughts as you work through assignments (and future work-related diagrams).

**From Logical DFD to Physical DFD - Getting Started**

**Content Author:** Terri Washburn, Ph.D.

And so we move on now, from our logical data flow diagram to a physical data flow diagram, which means we go from high-level, rather indistinct notions of the data sets, entities and data flows that are involved, to very project-specific and organization-specific physical DFDs. We will begin to add content and context to the original logical data flow that will direct the database and programming staff as to the exact needs of our project. That includes specifying types of files, file names, and other relevant details.

|  |  |
| --- | --- |
| Note that this is often a joint effort! Database experts and programmers are often brought in to consult on such diagrams for a couple reasons: it helps them to be involved early on and to avoid errors in logic or assumptions, and they are the ultimate experts of the existing databases and files for the organization. Analysts rarely have the technical expertise that database and programming experts do. | A picture containing text, person, computer  Description automatically generated |

Let's take our logical DFD on grading, below, and see what we can do with it as far as turning it into a physical DFD.

|  |  |
| --- | --- |
| Let's begin with this simple **use case** below (remember: DFDs naturally evolve from use cases). | Diagram  Description automatically generated |
| From this, we can construct the simple **logical DFD** you've already seen in previous lectures: | Diagram  Description automatically generated |

Not really any question about **what** we want to do here. It's the **how** we are thinking about now.

So let's put the physical data flow diagram together. To do that, click on the next link. Don't worry; I'll repeat the logical data flow there so that you can refer back to it right in context.

**From Logical DFD to Physical DFD - Getting Specific**

**Content Author:** Terri Washburn, Ph.D.

The **physical** data flow diagram will be similar in format to the **logical** DFD; however, it will be **more detailed** and **more specific** about '**how'** things will take place. Let's add some assumptions to the above scenario, and that is that:

* The student is submitting his/her assignment in Moodle;
* The professor accesses the assignment from within Moodle;
* The professor enters the grade in Moodle.

Keep in mind that **if** there are other scenarios - other computerized ways to submit, grade and view assignments- they would need to be incorporated into the physical DFD as well, or be a separate DFD altogether. But let's use this Moodle example because you all know Moodle, so it is **in context** for you.

Let's pretend that Moodle doesn't have a way to submit assignments and there is no way for the professor to grade them electronically and upload your score and feedback within Moodle. For purposes of this exercise this is our project: adding a computerized method of submitting and grading assignments.

Let's begin by separating the student from the computerized parts of this DFD. We do this with a dotted line. The dotted line represents that this is an entity outside the computerized part of the data flow, and is called a **human-machine boundary**. We know there is a **student** who is submitting the assignment and a **professor** who will be reviewing and grading the assignment. Both of them are outside the computerized parts of the DFD so we represent them thus:

|  |  |
| --- | --- |
| Diagram  Description automatically generated | The dotted lines represent the **human-machine boundary**. Ignore the small horizontal lines for now. They are part of the data flow, as we'll see in the next step. |

Next we begin to add details about the movement of the data, or the **data flow**. We provide more specifics about what each process and each entity (here, the student and the professor) are doing with the data.

|  |  |
| --- | --- |
| Diagram  Description automatically generated | We're not done yet. This is a partial physical DFD. But you can tell that the **student data flow** is submitting an assignment into the computerized process called "Upload Assignment" and the **professor will be requesting** an assignment - and more, as we will see next. |

What happens next, logically? You may think "the assignment will be graded" - but from a systematic standpoint, the assignment can't be graded unless it is stored somewhere that the professor can retrieve it. So we elaborate on the **data store** that was in our logical DFD and we give it a name and file type (here: a MYSQL table) and show the assignment being stored off:

Diagram

Description automatically generated

Keep in mind this is all the same physical DFD. I'm showing you portions of it, as we go along.

So the assignment is stored off, and next it needs to be graded. We know that from the logical data flow we worked on in the last lecture, remember? Note that the professor's steps are not recorded in the logical DFD (shown below), but that is okay because we are now expanding upon the logical data flow by providing more details.

*Our original logical DFD*:

Diagram

Description automatically generated

Before the professor can grade the assignment s/he must retrieve it. And s/he will do so from the same **data store** in which it was placed by the student's process:

Diagram

Description automatically generated

What does the professor do next? S/he grades the assignment, as shown in process 1.3:

So we will add the grading of the assignment and storing of the grades to our physical DFD:

Diagram

Description automatically generated

So take a look, once again, at the level of detail in the **logical data flow diagram**:

Graphical user interface, diagram

Description automatically generated

And compare that to the detail in the **physical data flow diagram**:

Diagram

Description automatically generated

Can you see how we have moved from simply **what** needs to be done to **how** it will be done? Can a programmer work from this? Yes, but there is more we can do to help them out, and we will discuss that in the next lecture.

**Content Author:** Terri Washburn, Ph.D.

As I mentioned in a previous lecture, the formal creation of data flow diagram symbols doesn't always take place - at least not right away. This article attests to that!

I recommend you read through this optional article (though you won't be tested on it) that says (in the author's words):

"On actual projects it's far more common just to stand at a whiteboard with one or more project stakeholders and simply sketch as we talk through a problem."

See more at: <http://www.agilemodeling.com/artifacts/dataFlowDiagram.htm#sthash.6tuU0Q0K.dpuf>

I recommend you read through it, as we will be doing some data flow diagramming and you will be tested upon it, and the more examples you familiarize yourself with, the better. Click the link above to read the article.

**Structure Charts**

**Content Author:** Terri Washburn, Ph.D.

As we move deeper into the design phase, we are getting closer to the time when the project will be handed over to developers or programmers to work on. As much as is feasibly possible, we want to make sure they will be carrying out the project specifications the way they are intended to be carried out so that the system operates as planned.

Though most analysts are not programmers (although some are, including your professor), there are some tools an analyst can use to communicate with the programmers in a not-so-highly technical manner. There are ways that the analyst can specify what he or she expects the program (or application or system) to do without getting into the actual programming language itself.

One of the ways they can do so is to use hierarchical charts or **structure charts**. Structure charts show the future programming code in a **hierarchical**manner and are intended to demonstrate:

* sequence
* selection
* iteration

Read about structure charts in your textbook as well as by **viewing the except below:**

[**Structure chart**](https://www.slideshare.net/arnoldindia/structure-chart)from [**Roy Antony Arnold G**](https://www.slideshare.net/arnoldindia)

If we were to consider doing a structure chart for our Moodle assignment example in the previous lectures, we would begin by **breaking down all the logical steps** in the process and create a tree-like structure (hierarchy) that identifies each step individually

For example, here is how we might illustrate:  
**How the Professor will Grade the Submitted Moodle Assignment**

|  |  |
| --- | --- |
| Diagram  Description automatically generated | Text  Description automatically generated |

Note that the above rendition is not the only way to create a hierarchical chart, and that the steps (1.1 through 1.4) and data flows are likely not all that would actually need to be given to the programmer, if he or she were to program / code from this example. But it gives you an idea of how the analyst can communicate with the programmer without using a programming language. It is also a very visual way of going over a process with a user, as well.

As mentioned earlier in this course, there are many tools and diagrams to help us through the systems development life cycle and on to the implementation of the project. The next tool we are going to discuss is called **pseudocode**. Pseudocode is a more textual method (actually a text-based method) of communicating specifications (directions) to the programming team.

# Pseudocode

**Content Author:** Terri Washburn, Ph.D.

What is **pseudocode**?

Most software programs are developed using a programming language, like C++ or Java. These languages have a specific syntax that must be adhered to when writing program's source code. Pseudocode, on the other hand, is not a programming language, but simply an informal way of describing a program. It does not require strict syntax, but instead serves as a general representation of a program's functions.

Retrieved on August 26, 2014, from <http://www.techterms.com/definition/pseudocode>

**Pseudocode** is a loosely-written way of telling a programmer how to write the program he or she is going to be completing for the project - or at least a part of the program.

The actual programming language itself does **not** have to be used, but the **logic** should be sound.

|  |  |
| --- | --- |
| From the same site as listed above comes this explanation: | Graphical user interface, text, application, chat or text message  Description automatically generated  (This is assuming that **"i"** is a data field that is keeping count of something). |

### Practical Scenario

|  |  |
| --- | --- |
|  | Assume in the example above **we are keeping track of nine entrants into a premium seating booth of a ball game**.  As people order tickets online:   1. a counter (a data variable we will call "i") is first set to zero, and 2. then incremented by 1 every time a seat is sold in that booth or box; 3. once the variable, "i" reaches 10, tickets for that booth can no longer be sold. |
| Therefore, the pseudocode example above could be expanded to say: | Graphical user interface, text, application  Description automatically generated |

Do you see that the pseudocode example is much closer in structure to English, and not PHP code, and thus can be written by (and understood by) analysts as well as programmers? It would be far simpler for someone to catch an error at this point ("wait - the box seats hold 19, not 9") than once it has gone into programming or testing or, worse yet, into production.

In the next link, take a few moments to watch a video on creating pseudocode. Then we are going to go back to our Moodle grading example and work through that in pseudocode, for the last lecture this week.

**Pseudocode - Part 2**

**Content Author:** Terri Washburn, Ph.D.

If you watched the previous link (the video tutorial) you have a good idea of how to create pseudocode. Let's now review this familiar-looking structure chart and draft up a bit of pseudocode to go with it.

Remember that the syntax of the language does not matter in pseudocode. Therefore, if I asked ten of you to write up pseudocode for this chart, I might get ten completely different versions. However, if I asked ten PHP programmers to do so, their syntax would be markedly similar. That is because **programming is concerned with syntax**, and **pseudocode is not**.

What is syntax, again? It is: [Click for answer](javascript:showAnswer('ansX');)

|  |  |
| --- | --- |
| Text, application, chat or text message  Description automatically generated | Diagram  Description automatically generated |
| Let's assume the professor is reading through all the assignments until no more remain in the assignment file: | Text  Description automatically generated |

What do you think? Is that readable enough for a programmer to understand what he/she must do?

I think so! We will work on pseudocode as part of class this week. Be prepared to do so!

## **Database And SQL Tip Sheet**

This is excerpted from the Internet, showing you the types of SQL statements you will need to be familiar  
with for the assignment this week.  Source: https://www.w3schools.com/sql/sql\_intro.asp

SQL is a standard language for accessing and manipulating databases. Pronounced either S-Q-L, or "sequel."

**What is SQL?**

* SQL stands for Structured Query Language
* SQL lets you access and manipulate databases
* SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

**What can SQL Do?**

* SQL can execute queries against a database
* SQL can retrieve data from a database
* SQL can insert records in a database
* SQL can update records in a database
* SQL can delete records from a database
* SQL can create new databases
* SQL can create new tables in a database
* SQL can create stored procedures in a database
* SQL can create views in a database
* SQL can set permissions on tables, procedures, and views

**SQL Syntax**

* The commands are CAPITALIZED.
* The fields (or columns) and tables (or database file names) are not.
* The SQL statement is ended with a semi-colon.
* There are many SQL commands. We are going to focus on SELECT.

**What is the SQL SELECT Statement?**

* The SELECT statement is used to select data from a database.
* Once it finds the requested data, it is stored in a result table, called the result-set.

**How to Format the SELECT Syntax:**

SELECT column1, column2, ...  
FROM table\_name;

Here, column1, column2, ... are the names of the columns (or fields) you want to select  
data from. So, if there is a field called "PatientName" or "CustomerlD" or "Birthdate" (and  
so on), you specify that exact spelling of the field/column. If you want to select all the  
fields available in the table, use the following syntax:

SELECT \* FROM table\_name;

Note: the\* means to return all fields in the table. Otherwise, put the field or column name(s) there. Also that the SQL commands (SELECT and FROM) are capitalized.

Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

| **Customers Table** | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Mark's Computer Shop | Mark Smith | 123 Main St. | Royal Oak | 48987 | USA |
| 2 | Jon's Motorbikes | Jon Moreno | Mataderos 2312 | Mexico D.F. | 05021 | Mexico |
| 3 | Ben's Barber Shop | Ben Jones | 789 Mill St. | Troy | 48321 | USA |
| 4 | Kat's Tattoos | Katherine Hardy | 012 Sun Ave. | Detroit | 48147 | USA |
| 5 | Amy's Deli | Amy Anders | 345 Tenth St. | Bloomfield | 48369 | USA |

The table above contains five records (one for each customer) and seven columns (CustomerID, CustomerName, ContactName, Address, City, PostalCode, and Country).

**SQL Statement Examples**

* The following SQL statement selects all the records in the "Customers" table:

SELECT \* FROM Customers;

* The following SQL statement selects just the "CustomerName" and "City" columns from the "Customers" table above:

SELECT CustomerName, City FROM Customers:

-Note the comma between the columns with commands in all CAPS

SELECT CustomerName FROM Customer:

-Note no comma needed unless you want multiple columns

The WHERE clause is used to select specific records. It is used to extract **only those records** that meet a condition that you specify.

WHERE Syntax

SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

Example: SELECT CustomerName FROM Customers WHERE Country='Mexico';

* Using the table above, the statement would return a result of: Jon's Motorbikes (only customer from Mexico)
* The condition can be an equal sign (=), greater than or equal to (>=), or any number of conditions that are appropriate to the field.
* Note that the **value of the field** you're looking for **must be encased in single quotation marks if it is text, or characters**.

**Some of the Most Important SQL Commands**

* + *SELECT - extracts data from a database*
  + *UPDATE - updates data in a database*
  + *DELETE - deletes data from a database*
  + *INSERT INTO - inserts new data into a database*
  + *CREATE DATABASE - creates a new database*
  + *ALTER DATABASE - modifies a database*
  + *CREATE TABLE - creates a new table*
  + *ALTER TABLE - modifies a table*
  + *DROP TABLE - deletes a table*
  + *CREATE INDEX - creates an index (search key)*
  + *DROP INDEX - deletes an index*

Students should know the following commands for the class assignment: SELECT, FROM, and WHERE