**CDJMM Software Solutions.**

Deliverable #4

CS131-02 Fall 2011

December 15, 2011

Contents

[1.0 Introduction 3](#_Toc311667508)

[1.1 Project Overview / Problem Statement 3](#_Toc311667509)

[1.2 Team Members, Roles and Signatures 3](#_Toc311667510)

[2.0 Revised Requirements 4](#_Toc311667511)

[2.1 Barcode Reader: 4](#_Toc311667512)

[3.0 Data Design 4](#_Toc311667513)

[3.1 ERD Diagram 4](#_Toc311667514)

[3.2 ERD Entities and Relationships 5](#_Toc311667515)

[3.3 Sequence Diagrams for the most Critical use cases 6](#_Toc311667516)

[3.3.1 Rent Equipment 6](#_Toc311667517)

[ 6](#_Toc311667518)

[3.3.2 End Rental 7](#_Toc311667519)

[3.3.4 Generate Report 8](#_Toc311667520)

[3.4 Data Dictionary 9](#_Toc311667521)

[4.0 Architecture Design 11](#_Toc311667522)

[5.0 Detailed Class Design 12](#_Toc311667523)

[5.1 Decision to Employ a Classless Design 12](#_Toc311667524)

[5.2 Customer Rental DFD – Level 0 13](#_Toc311667525)

[5.3 Customer Rental DFD – Level 1 14](#_Toc311667526)

[6.0 User Interface Design 18](#_Toc311667527)

[PHP Programming Tools 18](#_Toc311667528)

[8.0 Restrictions, Limitations, and Constraints 19](#_Toc311667529)

[9.0 References or Sources Used 20](#_Toc311667530)

# 1.0 Introduction

## 1.1 Project Overview / Problem Statement

The following document will detail the implementation of  a web-based software solution which meets the requirements of the Sacramento State Aquatic Center (SSAC). In particular, the system being described will effectively automate the customer registration system which had previously been implemented via a manual “paper and pen” process.

This document will outline, describe and illustrate all of the modules, classes, structures and other elements -- including their relationships and interactions -- which encompass the automatic solution that has been developed as a replacement for SSAC’s existing manual system of registration. From this point onward, the name of this automated system shall be the “Aquatic Center Tracking” system, or ACTS.

## 1.2 Team Members, Roles and Signatures

While each member had substantial contributions throughout each

phase of the development of the ACTS system, here are the formal roles

which were attributed to each individual:

Chiraag Thakor, Documents Writer/Project Reviewer

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

     Darby Hannon, Leader/Project Organizer/Manager

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

     Joubin Jabbari, System Coder/Manager/Interface Design

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

     Madelyn Bachiller, Documents Writer/Project Reviewer

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

     Mike Poku, Technical Writer/Object-Oriented Analysis

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# 2.0 Revised Requirements

## 2.1 Barcode Reader:

Due to insufficient time and also expense, the barcode reader and printer have been removed from the design and implementation. Initially, inclusion of a barcode reader was listed as a nonfunctional requirement. This barcode reader was supposed to ease and facilitate the process of “checking out” a customer that has returned from a rental trip. Rather than the staff or the customer having to locate information manually, the staff could potentially simply scan the wristband of a returning customer. This requirement, although it has been removed, could conceivably be added to ACTS at a later, more convenient time since -- being a nonfunctional requirement -- it’s removal does not violate the core purpose and main functions of the system.

# 3.0 Data Design

## 3.1 ERD Diagram

## 3.2 ERD Entities and Relationships

## 

## 3.3 Sequence Diagrams for the most Critical use cases

### 3.3.1 Rent Equipment

1. Enter number of people going on the trip
2. Rental Agreement Page: Waiver is show
   1. Enter Name(s)
   2. Select either Adult or Minor for each person
   3. Click on I/We Agree button
3. Next Page displays Non-Certified and Certified Equipment
   1. User enters quantity of the equipment to rent
   2. click Submit button
4. Enter names of who will go in which equipment and click Submit Button

### 

### 3.3.2 End Rental

1. Enter number of people going on the trip
2. Rental Agreement Page: Waiver is show
   1. Enter Name(s)
   2. Select either Adult or Minor for each person
   3. Click on I/We Agree button
3. Next Page displays Non-Certified and Certified Equipment
   1. User enters quantity of the equipment to rent
   2. click Submit button
4. Enter names of who will go in which equipment and click Submit Button

### 3.3.4 Generate Report

## 3.4 Data Dictionary

Person Table

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Type** |
| id | Identification Number of the Person | int |
| first\_Name | Customer’s First Name | var char |
| last\_Name | Customer’s Last Name | var char |
| phone\_Number | Customer’s Phone Number | var char |
| email\_address | Customer’s Email Address | var char |
| cert | What the Customer is certified for | int |
| person\_type | Admin or Staff or Customer | int |
| Adult or Minor | Customer’s Age (Adult if 18 and above or Minor if below 18) | bool |

**Inventory Table**

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Type** |
| id | Identification of the equipment Number | int |
| name | Name of the equipment | var char |
| quantity | How many equipment | int |
| description | Caption to describe the equipment | var char |
| cert\_required | Required certification for specific equipments | int |
| image\_address | Where the image is located for equipment | var char |
| reg\_price | Regular price per hour for the equipment | int |
| alumni\_price | Alumni price per hour for the equipment | int |
| special\_price1 | Special price per hour for the equipment | int |
| special\_price2 | Special price per hour for the equipment | int |
| capacity | Number of possible people for the equipment | int |
| weight\_limit | Weight limit of the equipment | int |

**Trip Table**

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Type** |
| id | Trip Identification Number | int |
| transaction\_id | Transaction Identification Number | int |
| p\_id | Person Identification Number | int |
| total\_price | Calculated total price of the trip | int |
| b\_paid | If the whole trip is already paid or not | bool |

**Transaction Table**

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Type** |
| transaction\_id | Transaction Identification Number | int |
| start\_Time | Starting Time of Rental per equipment | time stamp |
| end\_Time | Ending Time of Rental per equipment | time stamp |
| name | Name for the transaction | var char |
| total\_price | Calculated price for the transaction | int |
| b\_paid | If transaction is already paid or not | bool |
| equipment | Name of the equipment rented | var char |

# 4.0 Architecture Design

ACTS has the following class diagram, which shows all of the necessary relationships, associations and multiplicities that are involved in the interaction of  the major classes in the system. The diagram illustrates the main objects at play when the customer first logs in, and then all the way up until the customer checks out and generates a final point of sale.  
(in progress …..)

# Detailed Class Design

## Decision to Employ a Classless Design

There are many ways to conceptualize the solution to a programming problem, the most popular among them being an object-oriented approach employing a class-based design. However, after careful consideration we feel that although it may be possible to use an OO approach to achieve the goals of the ACT system, doing so would be become somewhat awkward and inelegant because of the nature of the project.

This project is very much web-based and rooted in the mechanics of PHP and HTML. We feel that it would be difficult to justify the existence of a class design based upon the lack of common properties and methods across the entities of the system. We do not want to force a class to generalize the behavior of disjoint entities in the system, which subsequently would only produce a class template with low cohesion.

It is true that there are some elements in the ACT system which would benefit from an OO approach (such as a Person class to represent the staff or customers). However, we feel that the system as a whole – particular with respect to how the entities interact and communicate – would be better analyzed using a different methodology. Therefore, rather than a class diagram we choose to represent our system via a Functional Modeling paradigm such as Data Flow Diagrams which clearly illustrate the actors, entities and how they flow and communicate within the context of a specific use case.

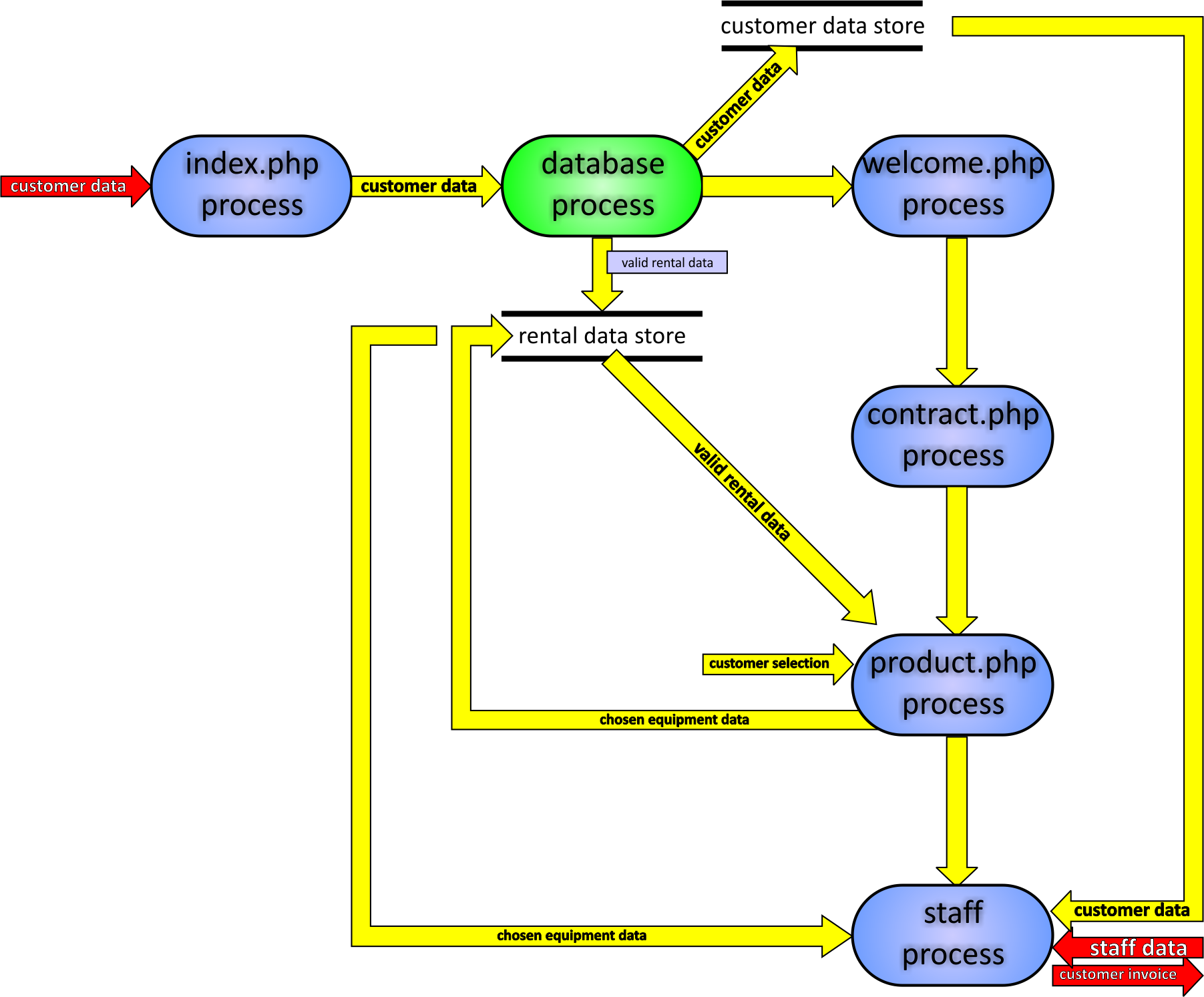
## Customer Rental DFD – Level 0

At the highest level of abstraction, our design of the ACT system is extremely simple when you consider the Level 0 model of the DFD, as is shown in the following figure:



The entire purpose of the system is for the system to stand as an automated intermediary between the customer and the staff personnel. The customer provides data to the system, and then the system processes that data and then ultimately provides the staff with a final invoice representing that customer’s rental transaction. During the process the staff may also input data into the system such as customer end time and so forth.

## Customer Rental DFD – Level 1

At the next level of abstraction, we break down further the “ACT System Process”, as shown:  


Within the ACT System Process, there are additional processes which are responsible for a number of actions within the system, such as – for example – the process titled “index.php” which is a php module. Conceptually, our design dictates that we view index.php as a singular entity which is responsible for three distinct tasks:

1. Displaying to the user a web-based Graphical User Interface.
2. Receiving input from the user through this GUI.
3. Reporting (outputting) the results of the user’s request to the database.

As the figure illustrates, once index.php receives the user’s data, it is forwarded to the database process. The database process is also responsible for several tasks:

1. Is the user a valid user? Verify.
2. Is the user a new or returning user?
3. Did the user provide valid security info (username, password, etc.)

When the database process verifies the validity of the customer, the database then queries its tables to generate a list of valid rentals based upon the credentials of the customer since some customers may or may not be eligible for some rentals. The database process then forwards this “valid rental data” to a data store we call “rental data store”. In addition, the database process also forwards the customer data to a store called “customer data store”. The data stores can be likened to computer memory; they can be easily accessed (read or write) at any time during the life of the transaction. Finally, the database process will spawn a new process called “welcome.php”.

Here too we view welcome.php as a singular process which takes in input and outputs a result. In this specific instance, the welcome.php process performs 3 specific tasks:

1. Displays the customer’s own information so customer sees he is logged in.
2. Display’s “class certifications” to customer so customer can review his own status.
3. Requests info from customer as to how many guests -- if any -- are with him.

Our welcome.php process receives information from the customer data store and then performs the three tasks outlined above. Next, welcome.php will spawn a new process called contract.php. Contract.php displays to the user all of the legal details, liabilities and waivers between the Aquatic Center and its customers. Contract.php requests from the user a simple response (a Boolean yes or no) as to whether or not the user complies with the conditions set forth by the contract. The response from the user is forwarded and saved in the customer data store. Contract.php then spawns the next process: product.php.

The product.php process takes as input the contract response which was saved in the customer data store. If the contract response is false, then at this point it is appropriate to inform the user that he cannot be serviced without agreeing to the contract and then we terminate. This specific use case is not shown. On the other hand, if the contract response is true, then at this point product.php will retrieve the valid rental data from the store and then based on this data it will populate its page with the corresponding products.

The product.php process performs the following tasks:

1. Retrieves valid rental data information from the rental data store.
2. Displays a GUI which illustrates the equipment available for rent by the customer.
3. Allows the user to choose specific equipment desired for rental.
4. Allows the user to choose specific quantities of the equipment desired for rental.

The user’s choices of equipment as well as their corresponding quantities represents a new set of data we call “chosen equipment data”. This data is saved in the rental data store.

At this point, from the Aquatic Center employee perspective we have everything that we need: We have the customer’s information in the customer data store and we have their chosen equipment data in the rental data store. Flow of execution then transfers to the process called: “staff process”.  
In the staff process, we have three inputs and one output:

1. [in] Chosen equipment data (saved in rental data store)
2. [in] Customer data (saved in customer data store)
3. [in] Staff data (provided by Aquatic Center Staff; includes staff info, customer end-time, etc.)
4. [out] Customer invoice (may include billing, time spent, reports, possible graphs, etc.)

The staff process reads the chosen equipment data from the rental data store and also the customer data from the customer data store. When the staff process receives its third input (the staff data which includes staff information, customer end-time, any possible discounts, etc.), the staff process will then begin grinding the necessary calculations to produce an invoice will include such things as billing (the final cost of service which the customer must pay), the amount of time the customer spent, an update of the type of equipment that was used and what has thus far been the most popular pieces of equipment, and a host of other reporting services which would be useful to either the staff or admin of the Aquatic Center.

This concludes the level 1 DFD design of the ACT system. The next section outlines the “Level 1 Pseudocode” which – based on the design just described – can be used as a roadmap toward the implementation of the ACT System Process.

# 6.0 User Interface Design

## PHP Programming Tools

    **PHP**: It is an opensource scripting language generally used in web development. This scripting language allows for the creation of dynamic and original web pages.

    **Mysql**: MySQL is an open source relational database management system. MySQL is a reliable and safe database management system used regularly in industry purposes. This platform provides for input of data from multiple users and can store information such as inventory which you can choose to have visible on your website. MySQL also allows for storage of customer information such as phone numbers and addresses for mailing which can be accessed on request.

# 8.0 Restrictions, Limitations, and Constraints

1. There was a limited time spent on this Software Design Document due to the project time constraint. A final product must be shown by December 15, 2011.
2. There was also a decision to use Data Flow Diagrams instead of Class Diagrams to better capture the flow of data since PHP and HTML were the programming tools that were used.
3. Due to insufficient time and also expense, the barcode reader and printer have been removed from the design and implementation.

# 9.0 References or Sources Used