Spring 2006

The Implementation of Time Management Solution

Robert M. Katz
Regis University

The Implementation of a Time Management Solution

Robert M. Katz
Regis University

A Project Report submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Information Technology

April 2006

Regis University
School for Professional Studies
MSCIT Program

Certification of Authorship of Professional Project Work

Submitted to: Joseph Gerber

Student's Name: Robert Katz

Date of Submission: April 17, 2006

Title of Submission: The Implementation of a Time Management Solution

Certification of Authorship: I hereby certify that I am the author of this document and that any assistance I received in its preparation is fully acknowledged and disclosed in the document. I have also cited all sources from which I obtained data, ideas, or words that are copied directly or paraphrased in the document. Sources are properly credited according to accepted standards for professional publications. I also certify that this paper was prepared by me for the purpose of partial fulfillment of requirements for the MSC 696 or the MSC 696B course.

Student's Signature: ____
Robert Katz

Regis University
School for Professional Studies
MSCIT Program

Advisor/MSC 696 and 696B Faculty Approval Form

Student's Name: Robert Katz

Professional Project Title: The Implementation of a Time Management Solution

Advisor's Declaration: I have advised this student through the Professional Project Process and approve of the final document as acceptable to be submitted as fulfillment of partial completion of requirements for the MSC 696 or MSC 696B course. The student has received project approval from the Advisory Board or the 696A faculty and has followed due process in the completion of the project and subsequent documentation.

Advisor

Hal Friskey
Name Signature Date

MSC 696B Faculty Approval

Joseph Gerber
Name Signature Date

Document Revision History

Change/Additions	Date
Front Matter	3/12/2006
Chapter 1	1/14/2006
Chapter 2	3/1/2006
Chapter 3	3/20/2006
Chapter 4	3/30/2006
Chapter 5	4/9/2006
Back Matter	4/14/2006
Final Formatting	4/16/2006

Acknowledgements

The author would like to thank his Wife and children who gave up so much of their time with him so he could pursue this degree. In addition, the author would like to thank all of his instructors who made this process so rewarding.

Abstract

This project proposes a technical solution that will improve the time tracking capabilities of the software department at Diagnostic Product Corporation (DPC). Currently, each developer submits ad-hoc monthly reports indicating how he or she spends their time. This is problematic for management since there is no standard methodology for recording time spent by each developer. Also, there is not a central repository for collecting this data, resulting in management not being able to accurately interpret how the staff is used from project to project. The goal of this project is the development of an application utilizing a networked database that would allow multiple user access to needed data, as well as reports that can be used by developers and management to interpret time spent. Implementing this system would give management the ability to better assess the needs of the department and the various projects within the department.

Table of Contents

List of Tables

List of Figures

Chapter One - Introduction

Problem Statement

Diagnostic Product Corporation (DPC) located In Flanders, New Jersey designs and manufactures immunoassay medical devices. The company devotes a significant portion of its annual revenue to the research and development of new technologies. The company has grown significantly since its inception expanding from six employees in 1989 to over six hundred employees in 2006. One side effect of this tremendous growth has been an increased complexity in the company managing its resources properly.

The Management of the software department at Diagnostic Product Corporation needs a better mechanism for tracking how staff personnel spend their time. The software department has grown tremendously over the past five years from a staff of six to forty. This rapid growth of the department has made it difficult for the Director of Software to explain to senior management how the department resources are allocated. The Director of Software needs a utility to help him 1) understand how current resources are being allocated across various projects and 2) provide supporting data necessary to justify to upper management the need to add additional resources to the department.

As other stakeholders were interviewed, it was discovered that additional problems could be resolved through the use of such a utility. The Manager of Software could use the data collected to analyze the amount of time developers were spending in each development phase. For example, what is the ratio of time spent in design versus code? This information could be used to guide the manager into possible areas that might be in need of process changes. Other questions this data could answer for the manager are how much time is spent in 1) meetings, 2) supporting other departments, and 3) sick time. Another problem that existed without this utility is there was no standard way of collecting this data.

Review of Existing Situation

Currently, each developer submits ad-hoc monthly reports indicating how he or she spends their time. This is problematic for management since there is not a standard method or format for recording the time spent by each developer. There is no standard format for these reports, therefore some developers use word documents while others submit Excel spreadsheets or send e-mails. Categories are not predefined. Each developer had his or her own categories. For example, one might allocate time to meetings while another would refer to this category as a Joint Application Design (JAD) session. Additionally, there is not a central repository for collecting this data. After receiving each developer’s report, the Manager of the software department would manually have to rollup the data into an overall summary for the department. This process takes several hours to complete and is highly interpretative due to the lack of standardization. The end result of the current process in use is that management cannot accurately interpret how the staff is used from project to project.

Goals of Project

The goal of this project is the development of an application utilizing a networked database that would allow multiple user access to needed data, as well as generating reports that can be used by both developers and management to interpret how time is spent. Implementing this system would give management the ability to better assess the needs of the department and the various projects within the department.

Finishing the application on time and meeting all requirements within budget will deem the project a success.

- Timeliness will be measured by finishing the project by the date promised in the project plan.
- Having the end-users and stakeholders complete a survey about the acceptance of the application will indicate if requirements were met.
- Budget compliance will be measured by not exceeding the time or resources allocated in the project plan.

Barriers and/or Issues

The development of this utility has a very limited budget. The only cost allowable for this project is the time and effort of the person developing the software. This puts certain constraints on the potential solution considered for this project. The development of the application is limited to being implemented in either Visual Basic 6 or C++. These are the only development languages currently being supported in the software department. A fixed budget also means that the database utilized cannot increase the cost of the project. No additional hardware or servers can be purchased to support this new application.

Acceptance of the new utility is another potential issue. It would definitely be problematic if the developers did not want to use the tool because it was not user friendly and intuitive. Developers might have a difficult time deviating from their current process. Therefore, user input into the design of this utility is of paramount importance. This would help to avoid creating a utility that would not be accepted by the end-users.

Scope of Project

The project will consist of creating the presentation, business logic, and data storage layers necessary to support the requirements defined for the Time Sheet Utility. The presentation layer will support the user interface, data input, report generation, and managerial/maintenance functions. The business logic layer shall support enforcing business rules to ensure the integrity of the data entered into the system. This tier will also provide for the projection and filtering of data necessary for the presentation layer. The data storage layer will be responsible for processing and storing data.

The project will allow for the developers to enter data on a weekly basis. The developers using the application will enter their time spent into a predefined list of categories. The utility will enforce all business rules. The utility will produce reports required by both management and the developer. The utility will maintain data for a minimum of 3 years. The utility will provide a way of ensuring all developers' enter their data in a timely manner and notify any developer who fails to provide their information. The utility shall provide a mechanism to add or delete tasks, as new ones are needed.

Definition of Terms

- Detailed requirements
  - This document contains what the application is required to do.
- Project plan
  - This document defines objectives and goals of the project. It specifies risks, assumptions, and constraints. It specifies key milestones for the project as well as necessary resources. It also defines the project team.
- Meeting minutes
  - Done throughout the project life cycle. These documents record discussions related to the various phases.
- User specifications
  - This document defines the graphical user interface. It contains mock screen prints. It also has flowcharts illustrating user interaction with the application to accomplish various tasks as defined in the requirements. It also states the purpose of any

controls on the screen, reports, and any necessary data validation.

- Entity relationship diagram
    - Defines the structure of the database, which includes entities with their attributes and the relationship between those entities.
- Technical specifications
    - This document is an output of the design phase. It defines how the system will be structured so it can accomplish what is required. It describes the various class and their methods including pseudo code of what should occur in each operation.
- Group
    - A hierarchical group of sub-groups. Some examples of groups are admin, project and support.
- Sub-Groups
    - A hierarchical group of Activities. Some examples of sub-groups are project names, answering question, and out of office.
- Activity
    - A hierarchical group sub activities. Some examples of activities are installations, sick time, and sub-projects.
- Sub-Activity
    - A category to enter task time against. This is the lowest level of the time tracking hierarchy. Some examples are specific software development phases like design, coding, testing.

- Task
    - An atomic unit of time spent on a specific sub-activity. A task is an intersection of a specific group, sub-group, activity and sub-activity. For example, a task may be 3 hours spent coding on the GUI layer of the Draco project. Here coding is the sub-activity, GUI development is the activity, Draco is the sub-group, and project is the group.

Summary

The Time Sheet Utility project will provide this author with a chance to apply all phases of both project management and software development. Project management techniques will be utilized to meet scope, time, and cost expectations. The software development cycle will be applied to increase the quality of the final deliverables so the end product is fit for use in meeting the requirements. It will be an important utility for the software department. It will help provide a mechanism for the management team to better understand how they are utilizing existing resources. This information will give insight into potential process improvements, resource needs, and standardization of reporting by the development team.

Chapter Two: Research & Methodology

Review of existing solutions available

According to Traylor (2006), existing solutions have both advantages and disadvantages associated with them. Existing solutions are attractive due to the fact that they are already developed, can be deployed immediately and have a fixed cost associated with them. However, existing solutions are not attractive because they are not built to meet the needs of the specific situation being addressed. However, most of the time you can find an *off the shelf* product that will meet most of your needs (Rubin, 2001). Prior to beginning the development of the Time Sheet Utility, various alternative solutions were considered. Other solutions considered included standardizing current reports submitted by staff personnel or buying an *off the shelf* application.

The first existing solution considered was to pick the best aspects of the current process and standardizing this format across the department. An example of the proposed format is listed in appendix A in the back of this paper. This approach would standardize the format used to submit time management data, but it did not meet all the requirements. It lacked the ability to easily see a summary data across all staff and/or projects. This was due to the fact that the data was not stored in a database format but was only collected via Excel spreadsheets. Another limitation with this approach was that it did not provide an easy way to add new categories to record time against. Because of these limitations, it was decided not to pursue using this approach.

The second existing solution considered was to buy rather than build. Various *off the shelf* applications were considered. One of the more attractive systems was called Time Sheet Express (http://www.timesheetxpress.com) (See Appendix B). It met most of the requirements as set out by the stakeholders. However, the lack of customization was a major drawback. Management also wanted to control the source code of the application so they could make future modifications as necessary. Lack of control and unreliability are potential pitfalls with using an *off the shelf* application (Newmann, 2002). Another deterrent to choosing this approach was its high priced that exceeded $4,445.00. However, this price included the initial cost and an additional lifetime upgrade fee.

Table 1 – Time Sheet Express Cost Analysis

Quantity	Timesheet Xpress Pricing	Price Per User (USD)
1	1 to 4 User Licenses	$59
2	5 to 9 User Licenses	$55.00 (7% Off)
3	10 to 24 User Licenses	$52.00 (12% Off)
4	25 User Site Licence	$47.80 (19% Off)
5	50 User Site Licence	$43.90 (25% Off)
6	100 User Site Licence	$38.99 (34% Off)
7	Lifetime Upgrade Protection	$45
8	Annual Upgrade Protection	$25

Cost Analysis

50 users x ($43.90 per user cost + $45.00 lifetime upgrade protection) = $4,445.00 (Overall purchase price)

Research methods

The two primary areas for research for this project were interviews and literature. Interviews were conducted with both managers and staff personnel to understand the type of issues and concerns this application should address. The interviews with management included the Director of the Software Department and the Manager of the Software Department. In addition, various software developers were interviewed including both application developers and team leads. These interviews provided insight into past experiences and how to approach the development of this application in an effort to avoid prior mistakes. In addition to interviews, another research method used was reading literature available in books and on the Internet related to software development. This research increased awareness of concepts and theories in computer science, which were most definitely applicable during the software development process. The combination of these research methods provided an adequate understanding of the concepts necessary to pursue the development of this application.

Contribution the project will make to the field

The Time Sheet Utility will provide an example of development done using the various concepts related to an object-oriented analysis and design. Many of the developers in the software department are new to object-oriented analysis and design. This application could be used as an example for other to emulate that have no prior experience with the concepts of object oriented analysis and design. It will show the validity of creating a logical distribution of responsibilities across a multi-tier architecture. Since completing this project, the application has already expanded beyond the original intent of being used by the software department. The quality assurance department has also begun to track the time spent by its employees using this utility.

The data collected by the utility has already been widely used by management to make decisions related to resource allocation and process improvements. It has provided management with the necessary raw statistics to make better-informed decisions about how costly a particular project or activity has been. Just this past January, additional headcount for the software department was based upon data collected using this utility.

Life cycle model to be followed

Iterative development is a technique for developing software that reduces risk by breaking a large body of work down into smaller, more manageable units of work. The iterative process handles change to the software system. This is possible since the iterative process provides the opportunity for full stakeholder involvement and continuous feedback with the software developer. The scope of the Time Sheet Utility project represents the first iteration of functionality desired by the stakeholders.

Traditionally, software development projects have been viewed as a collection of analysis, design, implementation and testing that result in a “big bang” release. This approach is known as the Waterfall Model of software development. According to Walton (2004) and Kroll (2004), this gives limited opportunities for the stakeholders to evaluate the project is on the right track.

Another methodology to managing a software project is known as the *Spiral* Model developed by Barry Boehm. The Spiral Model was designed to include the best features from the Waterfall and Prototyping Models, and introduces a new component – risk assessment. The term *spiral* is used to describe the process that is followed as the development of the system takes place. It basically consists of well-planned iterations. The goal is to build the high-risk items early in an effort to manage risk (Stiefel, 2005).

Iterative development is focused on iterations. Iterations could be viewed as mini-projects with specific objectives to be achieved. Some advantages to this approach are high risk items can be worked on first, and features having the highest business value may be deployed earlier. Iterative development anticipates change better than more traditional (Waterfall) models. Iteration milestones keep the developer continuously focused on results, which encourages peak productivity, and quicker progression to a functional system since each iteration produces a functional system (Kroll, 2004). Iterative development has four main phases. They are inception, elaboration, construction, and transition (Kroll, 2004).

The inception phase includes the activities necessary to define the project and get it underway. These activities include high-level requirements, project scope definition and planning. This phase of a project typically represents 5-10% of a project's duration (Kroll, 2004).

The elaboration phase includes the activities that set the stage for software construction activities. The activities in the elaboration phase include developing detailed requirements, conceptual models, establishing the software development environment, constructing prototypes to evaluate technical or presentation ideas and laying out the general implementation plan and schedule. The elaboration phase typically represents 10 –20% of a project's duration (Kroll, 2004).

The construction phase represents the core of the software development project, and it is here that the iterative nature of the project takes place. The general implementation plan is divided into iteration of 1 –2 months, during which discrete elements (features) of the software system are fully implemented. This is a key tenet of the iterative approach; by fully completing small components of the software product during iteration the stakeholders are able to deploy product incrementally (if so desired) to gain immediate return on investment. Fundamentally, the stakeholders are also able to modify the product's features in response to observations of prior iteration. They may even decide not to add additional features if not deemed necessary after the initial release. The construction phase typically represents 50 – 75% of a project's duration (Kroll, 2004).

The transition phase includes all of the activities necessary to bring a software development project to a successful conclusion. These activities include establishing the production operating environment, deployment, beta testing, and knowledge transfer. Sometimes this can occur intermixed with the construction phase to expedite use by the end user. The transition phase typically represents 10-20% of a project's duration (Kroll, 2004).

Development methodology

Design method

The application was divided into two main areas; each area requiring its own specific design method. The two areas of the application to be designed were the user interface/business rules layer and the persistent storage layer. The user interface/business rules layer is responsible for capturing and presenting information to the user. It is also responsible for enforcing all business rules. The persistent storage layer is responsible for maintaining data for later retrieval.

The design of the user interface and business rules layer would follow the principles of object oriented analysis and design. Requirements were collected to describe the problems the application should resolve. The first step in designing the system was to focus on breaking the system into a cooperating group of objects. The second step was identifying the interfaces for those objects. The third step was describing the interaction between these objects.

An object is a software package that contains a collection of related procedures and data. Within object-oriented programming, procedures are called methods, and the data elements are called properties. The reason for this approach is objects, which are instances of a class, make ideal software modules because you can define and maintain them independently of one another with each object forming a self contained, independent unit. The creation of this logical view of the system occurs during the analysis phase. In the design phase, the model is refined to take into account implementation considerations. The resulting physical view of the model captures the design decisions for later implementation. The object oriented design model assumes that during the application development life cycle, the analysis and design phases are traversed iteratively and incrementally, augmenting formal diagrams with informal documentation. The object oriented design methodology consists of accomplishing the follow steps during the modeling process: encapsulation, abstraction, assigning responsibilities, and relationships.

Encapsulation is the process of combining logically related procedures and data within a class/object. In this way, you insulate the object from the rest of the program. Because the object is using only data contained within it or passed to it, and the object executes only internal procedures, an encapsulated object is said to implement data hiding or information hiding. Said another way, encapsulation means that an object is not coupled to or dependent on any other object or procedure; instead, the object is independent and internally cohesive. The object does not contain any global or public variables and does not require any external procedures to execute its members. You can access and manipulate the data and behavior of an encapsulated object only through the object’s public interface (Fowler, 2000).

The use of encapsulation is beneficial in the design of the Time Sheet Utility in a variety of ways. First, it protects data from corruption by other objects or parts of the program. Second, it hides low-level, complex implementation details from the rest of the program and encourages data abstraction, which results in the capability to implement a simple public interface to a more complex set of private members. Third, it is easier to maintain legacy code or add new members to the object without affecting any procedures that currently call the object. Fourth, it makes debugging individual objects easier, as well as ensures that a bug within one object does not affect some other part of the system in an apparently unconnected way. Firth, encapsulation promotes reuse of the object by other programmers, thus improving their productivity.

Abstraction is a mechanism and practice to reduce and factor out details so that one can focus on a few concepts at a time. It means separating an object's behavior from its implementation. The human mind works better at problem solving when we can think in terms of abstraction. For example, a shape is an abstraction of pentagons, squares, and triangles. This lets the designer of a system understand what is common rather than focusing on specifics. It also creates a common interface that can be shared across various specific types. Abstraction is what leads to polymorphism in object-oriented development. The Time Sheet Utility benefits from abstraction by helping the developer to focus on the essential characteristics of the abstracted classes rather than their implementation details. Abstraction hides low-level, complex implementation details from the rest of the program.

Responsibilities refer to assigning roles to each component within the system. A major proponent of responsibility driven design is Rebecca Wirfs-

Brock. She believed in emphasizing the encapsulation of both the structure and behavior of objects. (Puttick & Tkach, 44) The goal with responsibility assignment is high cohesion. This means related tasks that the system needs to perform are assigned logically. Proper responsibility assignment leads to cohesiveness since each object in the system handles all tasks required of that entity. This leads to a maintainable system because responsibilities are not duplicated. For example, an object representing a printer is responsible for all aspects of what a printer should do such as maintaining its queue and sending a print job to the physical printer.

Relationships between the various business objects refer to establishing how the pieces work together as a system. Each part of the system must interact with other parts of the system. The output of this step is the creation of a class diagram. Software objects interact with each other by sending messages requesting that methods be carried out or properties be set or returned. A message is simply the name of an object followed by the name of one of its members. A message can have three parts; name of the receiving object, name of the object’s member (method or property), and values passed into that procedure (parameters). There are various types of relationships that can exist between objects within the system. The various types of relationships are specialization (inheritance), association and aggregation. Specialization represents an *is a* relationship where one object is a type of another object (Fowler, 2000). A classic example of this was presented earlier stating a shape *is a* pentagon. Association is where one object interacts with another object. Association represents a *uses* relationship. For example, a business object might use a data access object to persist its state to the database. Aggregation represents a *has a* relationship (Fowler, 2000). This is where one object contains another object. For example a classroom contains students or an airport contains airplanes. There are various degrees of aggregation depending upon how strongly the objects are coupled. The Time Sheet Utility illustrates many relationships between the objects contained in the application domain. For example, time is tracked by programmer on a weekly basis and each week contains a collection of tasks and a collection of tasks contains individual tasks.

The design of the data tier required slightly different design methods than the presentation/business logic layer. The design goal of the database was to build the persistent data structure in accordance with the process of normalization. According to Wikipedia:

> Database normalization is a process of eliminating duplicated data in a relational database. The key idea is to store data in one location, and provide links to it wherever needed. Well-normalized databases have a design that reflects the true dependencies between tracked quantities, allowing quick updates to data with little risk of introducing inconsistencies (2006).

The normalization process would be applied until 3rd normal form was achieved, then denormalization would begin as deemed necessary to support business objectives related to performance.

As previously mentioned, normalization is the process of minimizing data redundancy. All together there are six normal forms. Each of these normal forms includes all the characteristics of its predecessors and then adds additional rules. Therefore a schema that is in 3rd normal form must be in 2nd normal form. While it might seem the higher the normal form the better the database design, this is not quite true. There are other considerations besides reducing redundancy which make implementing the highest levels of normal form impracticable. Each individual situation must be examined on its own merit. For example, a high degree of normalization, say 4th normal form would probably result in numerous joins between entities. These additional joins would increase the time it would take the application to retrieve data. The degradation in retrieval time might outweigh the benefit of reduced redundancy. Considering this the design goal for the Time Sheet Utility database was to achieve 3rd normal form and not pursue the higher normal forms.

First normal form is very simple. A table scheme is said to be in first normal form (1NF) if the attribute value is atomic, there are no repeating groups, and all attributes are dependent on the primary key (Goren, 2004). An atomic attribute means it contains one piece of information. For example, Figure 1 illustrates an address attribute that contains the entire address including city, state and zip. This is not atomic. This address attribute should be broken up into separate attributes titled city, state, and zip. Figure 2 shows how to correctly represent the address and name attributes to satisfy 1st normal form. A repeating group means various attributes are additional instances of the same type of information. For example, an insurance policy table might define attributes such as car1, car2, and car3 to represent a multiple car policy. This is a repeating group since all three attributes represent the same thing.

Not Normalized

ID	Name	Address
3	Ben Goren	Post Office Box 964, Tempe, Arizona 85280-0964
4	Jane Doe	1234 Main Street, Mesa, Arizona 85345

Figure 1 – First Normal Form Bad Example (Goren, 2004)

Normalized

First Name	Last Name	Street Address	City	State	ZIP
Ben	Goren	Post Office Box 964	Tempe	Arizona	85280-0964
Jane	Doe	1234 Main Street	Mesa	Arizona	85345

Figure 2 – First Normal Form Good Example (Goren, 2004)

Second normal form is achieved if all of the strictly informational attributes (those not belong to any key) are attributes of the entity in the table scheme, and not of some other entity. For example, Figure 3 illustrates a violation of 2nd normal form. The attribute representing a spouse’s birthday should not be part of the people entity since the birthday is not related to the person. Figure 4 demonstrates how to relate the spouse’s birthday to their partner located on the person entity. Also, a table in 2nd normal form includes no partial dependencies (where an attribute is dependent on only a part of a primary key) (Goren, 2004). This aspect of 2nd normal form is only applicable when you have a composite primary key. For example, an entity called EmployeeCompany, that represents the relationship between employees and their employers, might have a composite key of Employee ID and Company ID. This entity might have a Wage attribute that represents an employee’s pay. This violates 2nd normal form since the Wage attribute only depends on the employee id and not the company ID.

Not Normalized

ID	First Name	Last Name	Spouses Birthday
3	John	Smith	02/07/65
4	Sue	Jones	07/26/76

Figure 3 – Second Normal Form Bad Example (Goren, 2004)

Normalized

Table: People				Table: Spouses	
ID*	First	Last	Birthday	Husband*-	Wife*-

	Name	**Name**	
3	John	Smith	08/01/62
4	Sue	Jones	10/12/75
5	Mary	Smith	02/07/65

>People:ID	>People:ID
3	5

Figure 4 – Second Normal Form Good Example (Goren, 2004)

Third normal form (3NF) requires 2NF be achieved and the entity contains no transitive dependencies (where a non-key attribute is dependent on another non-key attribute (Goren, 2004). This is similar to the partial key dependency described as part of 2nd normal form except instead of an attribute depending upon part of a composite key it depends on another attribute in the entity that is not a part of the entity’s primary key. For example, Figure 5 illustrates a violation of 3rd normal form. The employee entity has a non-key attribute called job ID, which represents the employee’s job. The employee entity also has another attribute called job title. Job title is dependent on the job id and not the employee id. Job Title violates 3rd normal form and should not be part of the employee table (Roman, 1997). Figure 6 shows how to correct this violation by creating a table of jobs and relating back to the employees table using the job id.

Not Normalized

ID	First Name	Last Name	Job_ID	Job Title
3	John	Smith	1	Salesman
4	Sue	Jones	1	Salesman

Figure 5 – Third Normal Form Bad Example

Normalized

Table: Employees			
ID*	**First Name**	**Last Name**	**Job_ID**
3	John	Smith	1
4	Sue	Jones	1

Table: Jobs	
Job_ID	**Title**
1	Salesman

Figure 6 – Third Normal Form Good Example

Implementation method

The Time Sheet Utility will implement a two-tier client/server architecture. The two-tier client/server architecture includes one or more client computers connected to one or more servers via a network. This approach would be beneficial for the application for a variety of reasons. First, it distributes the application processing between the client and the server. The client handles complex input validations, sorting data, and presenting the graphical user interface. The server processes the requests the client sends. For example, the database processes the query requests, which the client sends and returns the results back to the client. Second, it allows for central processing on the server. This supports the application requirement for a centralized data repository.

The two-tier architecture does present some disadvantages. Diagnosing problems is difficult because three potential bottlenecks could exist: the client, the network, or the server. Deployment and maintenance can also be an ordeal. A simple change to the application would require redeploying the application to all the client computers. However, in this scenario, this risk was mitigated by the fact that all of the end-users for this application were located in the same facility.

The development language for implementing the user interface and business components will be using Visual Basic 6, an object based language. The Visual Basic 6 language is referred to as object based because it does not support implementation inheritance. However, it does support interface inheritance. This will create some design constraints but it is a limitation being imposed on the development of the system since the only choices available for development are either Visual Basic 6 or C++. The decision to use Visual Basic 6 is based on the developer’s experience with it over Visual C++.

The implementation of the database will be accomplished using a relational database system. The database management system for the Time Sheet Utility will be Microsoft Access. This is due to the fact it supports all necessary design elements and is freely accessible to the end users of the system. In addition, all end users have network access to the Microsoft Access database and there is no cost associated with using Microsoft Access as the backend.

Review of the deliverables

Requirements Documentation

- Requirements will be generated indicating the functionality required for the application. One set of requirements will be created for the application layer (GUI) and another set of requirements will be written for the persistent storage layer (database).

Design Documentation

- Technical specifications will be generated to document the intended design of the system. The technical specification will document all classes, their properties, methods and events and the pseudo code contained within each procedure.
- User specifications will be created to illustrate the look and feel of the graphical user interface.
- Class diagrams will be generated to visualize the relationships between the various classes contained in the application domain.
- An entity relationship diagram will be created as part of the design of the persistent storage layer (database).

Implementation

- Source code of the application

- Database containing all tables and queries

Deployment

- Compiled executable and associated dynamic linked libraries
- Installation program

Resource requirements

The completion of this project will require software, hardware, and manpower resources. The software required for this project will consists of Visual Studio 6, Microsoft Access, and Microsoft Word. Visual Studio 6 will be used to complete the implementation of the application. Microsoft Access will be used to implement the database. Microsoft Word will be used for report generation. Hardware resources required are one server on which to locate the database. The application will be deployed to each client workstation. The human resources required to do this project will be one developer for a period of 62 workdays.

Outcomes

The expected outcome of the project is the successful deployment of the Time Sheet Utility to each of the developers in the software department at DPC. The application would be used on a daily basis to collect data about the how developers are spending their workdays. The collection of this data, in a central repository, would enable management to produce metrics, which would enable them to make better business related decisions. In addition, the data collected would have more value since the method of collecting the data would be standardized across all developers.

The Time Sheet Utility would reduce the effort spent to collect and process statistical data on how resources are allocated. The developers would spend less time to enter their information and management would spend less time to interpret the data. From a development standpoint, the Time Sheet Utility would be influential on other developers within the software department. It would influence future application design both at the application and database levels.

Summary

Research into the development of Windows applications has led this author to believe the best software development lifecycle to follow during the development of the Time Sheet Utility is the iterative method. This method of software development allows for the inevitable changes that come during the development of any software application.

An application's success cannot be based solely on following a specific software development lifecycle. In addition to this there must be a correct design and then the successful implementation of that design. Research into current design methodologies has led the author to the conclusion proper application design requires the application of object-oriented concepts such as abstraction and encapsulation. Database design is most effective when one applies the practice of normalization according to the theory espoused by Dr. Codd. Creating the correct application and database designs are major goals of this project. It is believed this will increase the maintainability, scalability, and usability of the Time Sheet Utility.

Additional goals of the project, besides producing the application, include producing other necessary documentation such as requirements, technical specification, and various other design documents. It is believed these deliverables are valuable in assisting other developers who have to maintain or enhance the application. The end result of producing this application is it would play a significant role in enabling Management to effectively monitor how human resources are used, thereby allowing them to make more enlighten decisions.

Chapter Three - Project Analysis & Design

Business Requirements

The software department is going to use this new application to replace the need to submit time sheets created in Excel to management each week. This application will collect data from each member of the software department. The data will reflect the breakdown of time spent on various activities. Requirements were gathered for both regular users and administrative users. All programmers are regular users and management is administrative users. Members of the software department and management will use this data to generate various outputs.

Standard User Requirements

- User shall be able to record actual time (in hours) spent on activities.
- User shall have ability to enter data on a daily basis.
- User time spent shall be rounded to the nearest ¼ hr increment.
- User shall have the ability to append comments to each activity they create.
- User's choices will be limited to choosing only activities and projects authorized by the Administrator.
- User can only enter their data.
- User can edit previously entered data.
- User can delete previously entered data.

- User cannot add, delete or modify descriptions of activities, projects or any other supporting data.
- User shall have the ability to navigate application without using a mouse.
- User shall have ability to look at previously entered data.
- User will not be able to see other user's data.
- User shall receive a reminder to enter data when they are late.
- User shall have ability to filter for records within a specific date range.
- User shall be able to produce output matching chosen dates, projects, and/or activities
- User shall have ability to group their data by sub-group and group levels.
- User shall have indication of how many hours they have entered for current day and week
- User shall be able to produce output in text format.
- User shall be able to produce output in Excel format.

<u>Administrative (Admin) User Requirements</u>

- Admin user shall have all input capabilities of a standard user.
- Admin user shall have ability to indicate if comments are required for a specific activity.
- Admin user shall have ability to add, delete or modify activities.
- Admin user shall have ability to add, delete or modify users
- Admin user shall have ability to add, delete or modify projects.

- Admin user shall have ability to limit access to system.
- Admin user shall be able to inactivate users, activities, and projects.
- Admin user shall have all output capabilities of a standard user.
- Admin user shall receive notification when user is more than one week late in recording data.
- Admin user shall be able to produce output indicating who has not complied in entering data within a week of when it is due.
- Admin user shall be able to produce output indicating when specific activities took place.
- Admin user shall have ability to view any user's data.
- Admin user shall have ability to roll up data in a logical manner.
- Admin user shall have ability to run ad-hoc reports based on any combination of projects, activities, users and date range.
- Admin user shall have ability to produce output so that it can be used as input into Excel.
- Admin users shall have ability to audit exception data to ensure quality of data. Exception conditions are as follows:
    - User records over 60 hours in a single week.
    - User with < 40 hours in a single week.
    - User records weekend activity.
    - User records over 12 hours on a single day.

Application requirements

The software department is going to use this new application to replace the need to submit time sheets created in Excel to management each week. The application’s front end will be designed in Visual Basic 6.0. The application’s back end will be a Microsoft Access database located on server that can be accessed by all end users. This application will collect data from each member of the software department. The data will reflect the breakdown of time spent on various activities. Members of the software department and management will use this data to generate various outputs.

Database Requirements

This database will be the central repository of all data collected from the users.

- The database will be newly created for this application.
- The database will contain tables, queries and reports that the front end will connect to in order to save, edit, delete data; or produce output
- The database will be password protected to avoid unauthorized access.
- The database will contain normalized data.
- One or more personnel who shall act as the database administrators shall maintain all supporting tables. Users will not be able to make additions or deletions to any supporting tables.
- Referential integrity will be enforced by set relationships within the database.

User Interface Requirements

- Menu items and buttons will have a letter underlined to indicate which key may be used to access it.

Technical design specifications

GUI Layout

The following flowcharts illustrate how the user can navigate thru the application. This section also provides screen captures of the graphical user interface. Figure 7 shows how the user would navigate from starting the application to the main screen. Upon starting the application, the user would be presented with a login screen. If the user logs in successfully the user would navigate to the main screen. The screen would default to filling with the user's most recent week of data. If no data exists, the user would be presented with a main screen containing no data. At this point the user can do a variety of activities from entering new data, editing existing data or navigating to another part of the application. Depending upon their security level, they may not be able to access all parts of the application. For example, only a user with admin rights can access the admin menu.

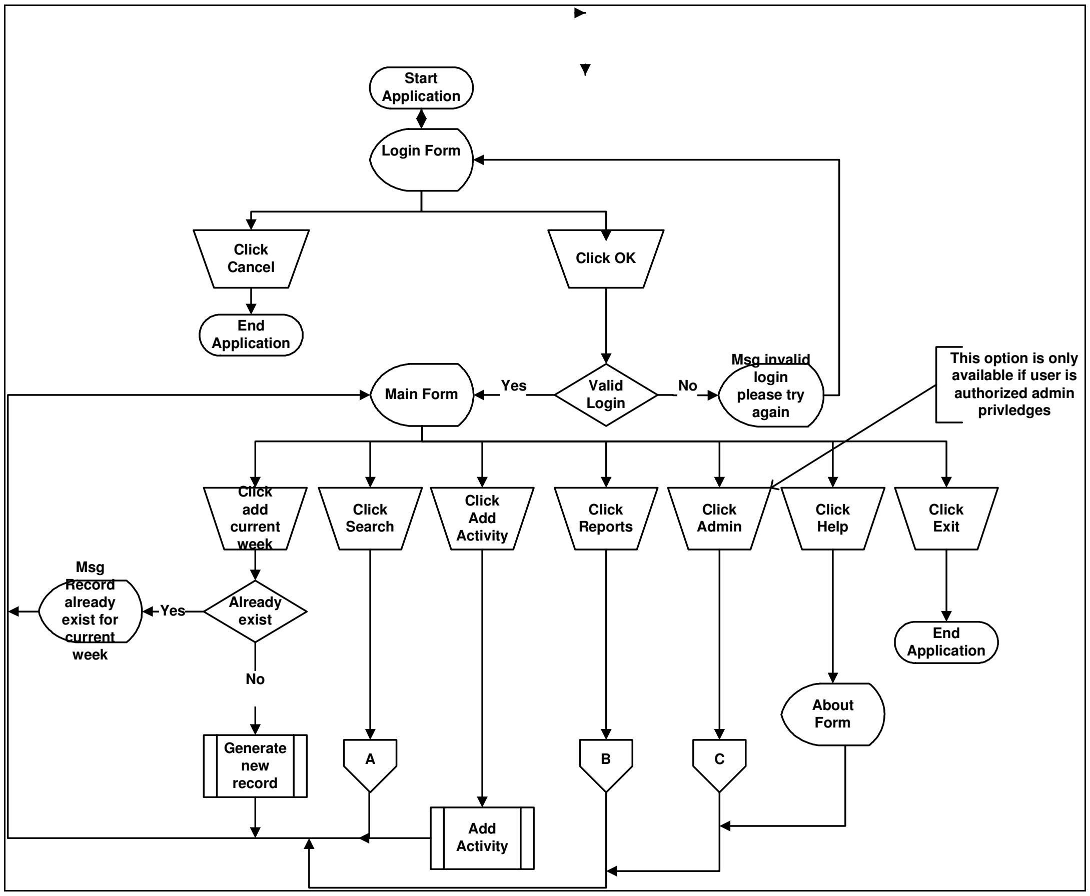


Figure 7 – Main Screen Navigation

Figure 8 provides a visual of the login screen. The user is required to provide a valid user name and password prior to being granted access to the application.

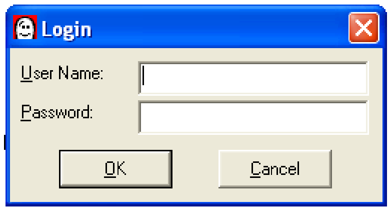
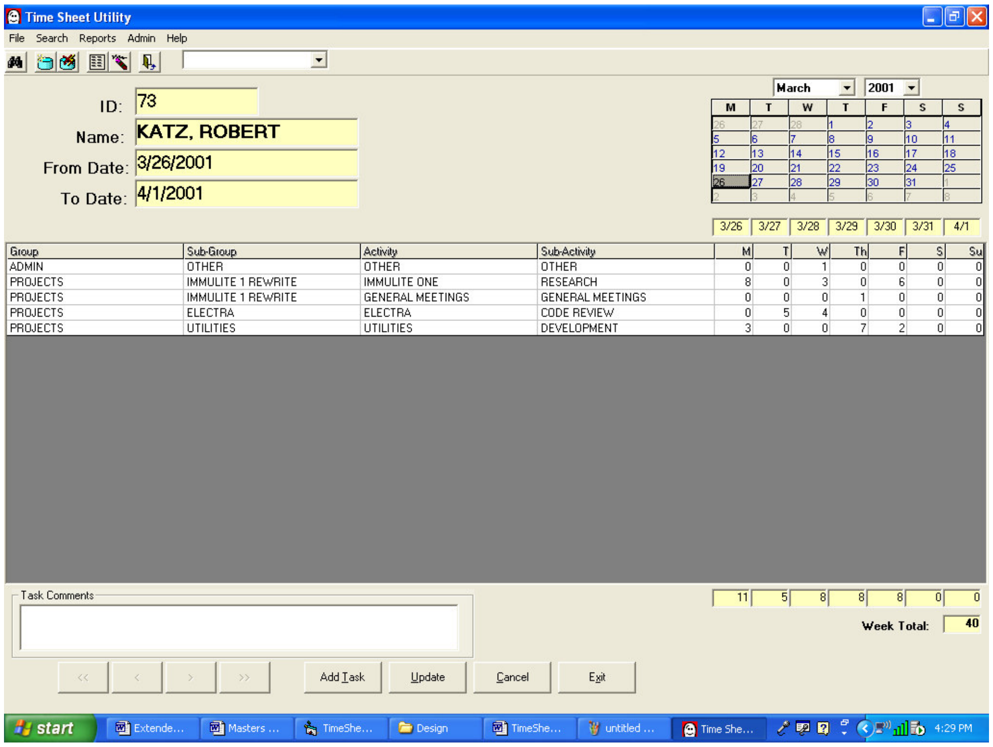


Figure 8 – Login Screen

Figure 9 presents a screen capture of the main screen of the application. The user will spend the greatest majority of their time at this screen. It is from here the user can choose what data they want to see, edit, add, or delete.

Users are presented with a calendar that gives them the ability to switch from week to week easily. Also the grid contains the collection of tasks entered for the currently selected week. All user choices in the grid are limited to valid entries based upon presenting the user with appropriate lists of choices. The user cannot type in any category. The user can attach comments to any task in the text field located at the bottom of the screen. The user is also presented with summary information containing the total hours spent for each day and the overall week. This summary information is used to give a visual indication of how many hours have been entered so far.

The drop down list box, located on the toolbar, is visible for admin-level users only and allows the user to select a different programmer. Once a different programmer is selected, the application loads with that person's data.



ID: 73
Name: KATZ, ROBERT
From Date: 3/26/2001
To Date: 4/1/2001

M	T	W	T	F	S	S
26	27	28	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Group	Sub-Group	Activity	Sub-Activity	M	T	W	Th	F	S	Su
				3/26	3/27	3/28	3/29	3/30	3/31	4/1
ADMIN	OTHER	OTHER	OTHER	0	0	1	0	0	0	0
PROJECTS	IMMULITE 1 REWRITE	IMMULITE ONE	RESEARCH	8	0	3	0	6	0	0
PROJECTS	IMMULITE 1 REWRITE	GENERAL MEETINGS	GENERAL MEETINGS	0	0	0	1	0	0	0
PROJECTS	ELECTRA	ELECTRA	CODE REVIEW	0	5	4	0	0	0	0
PROJECTS	UTILITIES	UTILITIES	DEVELOPMENT	3	0	0	7	2	0	0
				11	5	8	8	8	0	0

Week Total: 40

Figure 9 – Main Screen

Figure 10 depicts the user's interaction with the Search screen. This screen is used to search for tasks data that falls between the *from* and *to* dates provided by the user. This screen validates the user input and provides an informational message if any input is invalid or if no data matching the search criteria exists. If matching tasks are found the user is returned to the main form loaded with the data matching the chosen criteria.

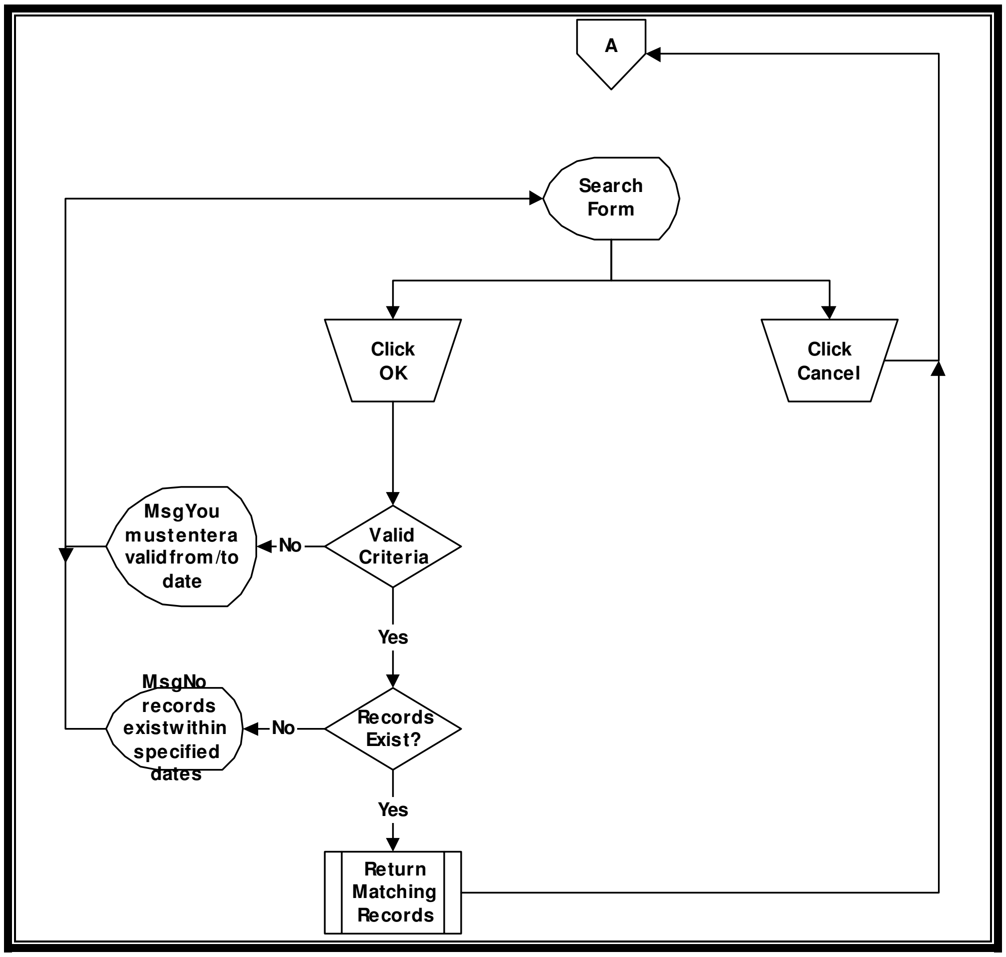


Figure 10 – Search Screen Navigation

Figure 11 provides a screen capture of the search screen. This screen allows the user to limit the tasks displayed to a specific date range. It contains validation logic to ensure valid data is entered in the two text boxes.

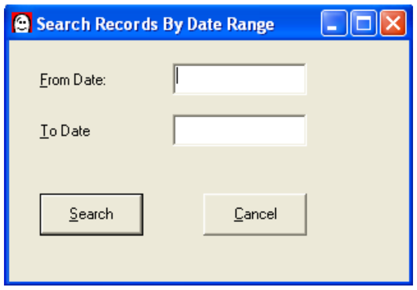


Figure 11 – Search Screen

Figure 12 depicts the user interaction with the report criteria screen. This screen is used to provide necessary criteria for use in report generation. The criteria provided include *from* and *to* dates and report format (rich text, HTML, or excel). This screen validates the user input and provides an informational message if any input is invalid or if no data matching the search criteria exists. If matching tasks are found the user is presented with a report in the chosen format containing tasks that fall between the selected dates.

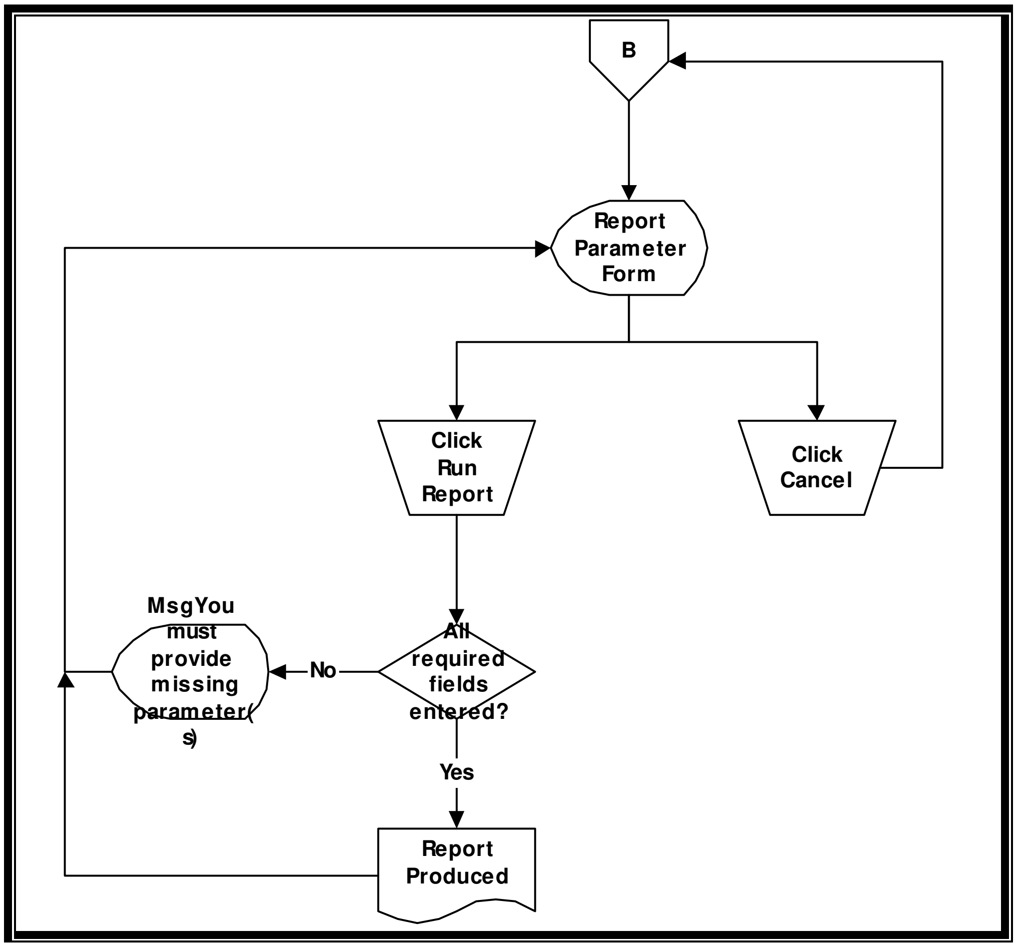
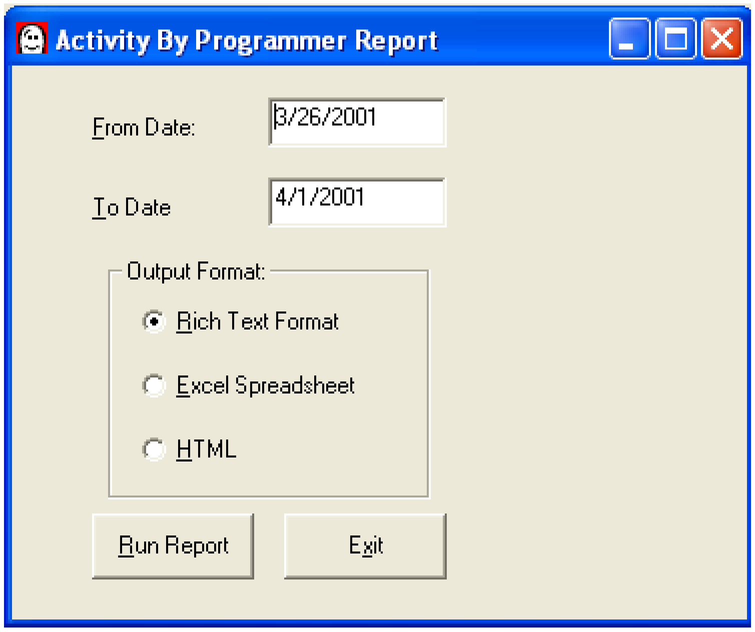


Figure 12 – Report Criteria Screen Navigation

Figure 13 provides a screen capture of the report criteria screen. This allows the user to run various reports. In this screen shot, the user can run an activity report for a chosen time range and output in a chosen format.



From Date: 3/26/2001

To Date 4/1/2001

Output Format: [✓ Rich Text Format] Excel Spreadsheet HTML

Figure 13 – Report Criteria Screen

Figure 14 depicts the administrative capabilities of the application. An administrative user can perform various system administrative tasks not accessible to the standard user. These tasks include maintaining system tables, inserting, updating, and deleting existing programmers, groups, subgroups, activities, and sub activities. An admin user can also perform database maintenance from this part of the application by choosing to compact and repair the database. This action resets all indexes and compact's the database file.

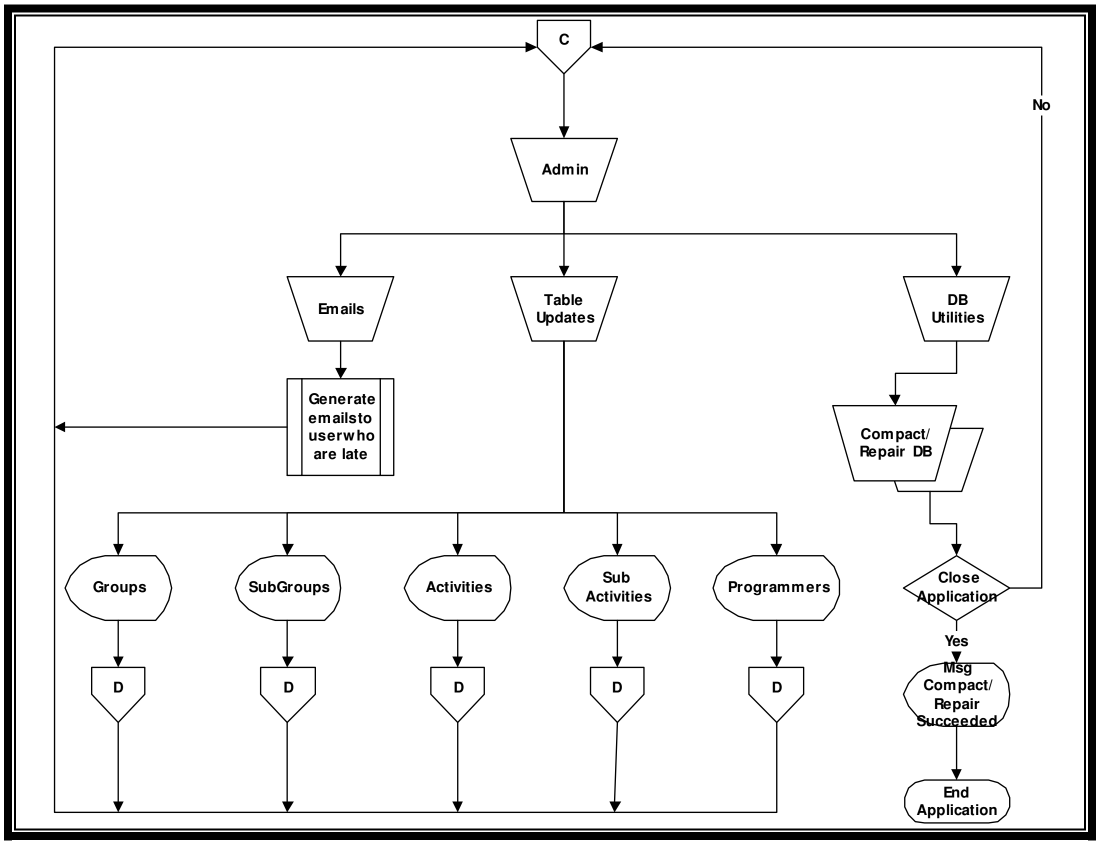


Figure 14 – Administration Menu Navigation

Figure 15 depicts the administrative capability of updating system database tables. This includes inserting, updating, and deleting existing programmers, groups, subgroups, activities, and sub activities. The screen performs all required screen validation to ensure no data anomalies occur. For example, the user cannot delete a programmer from the database if they have related records in the database. However, the user could deactivate the programmer so they can no longer access the system and will not appear on any lists.

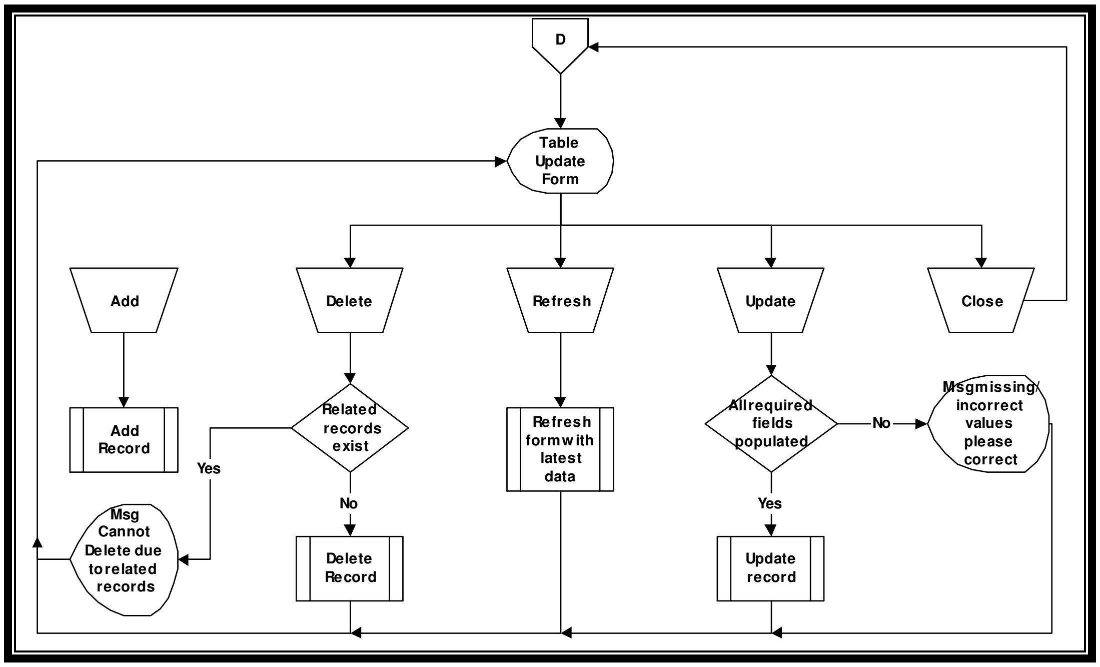
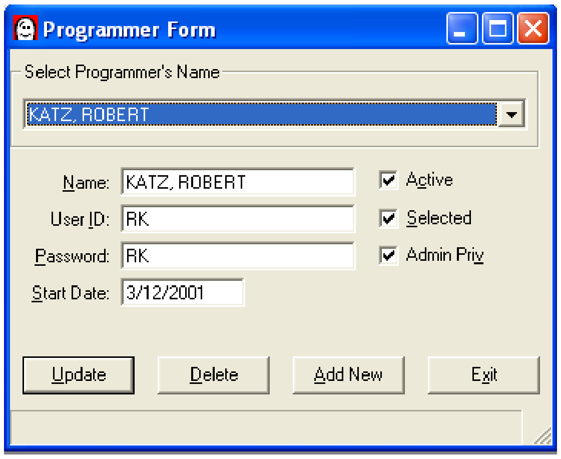

D
Table Update Form
Add
Delete
Refresh
Update
Close
Add Record
Related records exist
Yes
No
Msg Cannot Delete due to related records
Delete Record
Refresh form with latest data
All required fields populated
No
Yes
Msg missing/ incorrect values please correct
Update record

Figure 15 – Table Update Screen Navigation

Figure 16 shows an example of one of the table update screens. This screen is used to add, delete, and modify programmers.


Programmer Form

Select Programmer's Name: KATZ, ROBERT

Name: KATZ, ROBERT
User ID: RK
Password: RK
Start Date: 3/12/2001

☑ Active
☑ Selected
☑ Admin Priv

Figure 16 – Programmer Screen

Class Diagrams

Class diagrams are another output of the design phase. Building a class model involves identifying the classes that should exist in the system and how they relate to each other. This is a key process in designing an object-oriented system. A class diagram and the model it represents are useful because 1) it lends itself to building a system quickly that meets user requirements and 2) it helps promote building a system that will be easy to maintain and adapt to future requirements and 3) it provides future developer's of the application with a visual representation of the otherwise complex system. A class diagram can be used by software development teams similarly to the way an architect uses blueprints to communication his vision to the members' of a construction team.

These goals are important since proper design leads to a well thought out system that avoids redundancies in code through proper division of responsibilities and delegation. The class diagrams created as part of the design of this application helped in meeting the objective of building a loosely coupled tightly cohesive system.

Figure 17 shows a very high level view of the objects aggregated by the main screen. You will recall the main screen of the application is where the developer enters his task activity (See Figure 9). The goal of this design was to not do all of the work in the event handlers at the form level, but to delegate work to business objects that would encapsulate the business rules. For example, when the main screen is displayed it calls upon the CWeeks class to get the most recent week of data for the current user by calling the GetMostRecentWeekForProg method on the CWeeks class. The CWeeks class contains a collection of Week objects. Each Week object represents one week of data for a specific developer. Each Week contains a collection of objects representing individual tasks.

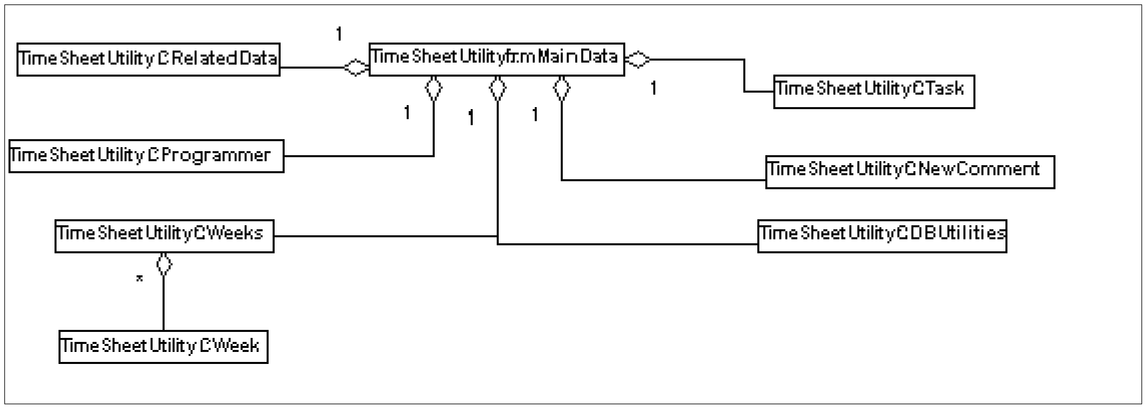

TimeSheetUtility.CRelatedData
TimeSheetUtility.frmMainData
TimeSheetUtility.CTask
TimeSheetUtility.CProgrammer
TimeSheetUtility.CNewComment
TimeSheetUtility.CWeeks
TimeSheetUtility.CDBUtilities
TimeSheetUtility.CWeek
1
1
1
1
1
*

Figure 17 – frmMainData Class Diagram

Figure 18 depicts the class diagram for the login screen of the application. Notice the usage of aggregation between the frmLogin (GUI), the CProgrammer (business object), and the CDBProgrammer (data object) classes. This was a typical design pattern used throughout the application. The goal was the logical separation of responsibilities between the GUI, business rules, and data access into different layers. Each layer has specific responsibilities. The form is responsible for collecting and presenting information. The business object enforces business rules and domain logic. Finally, the database access layer saves and retrieves information from the database. The basis for this design is it makes it possible to separate these layers and deploy them in a distributed manner. Also, it follows the object-oriented best practices of encapsulation and abstraction.

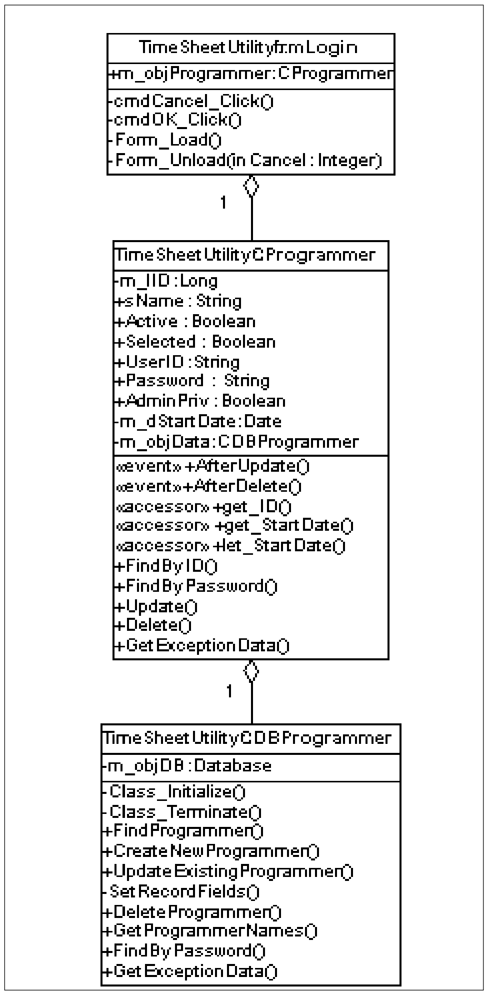


Figure 18 - Form Login (Class Diagram)

Figure 19 depicts the relationship between the classes utilized by the *Search Screen*. The key design concept illustrated by this diagram is the hierarchical relationship between many of the objects. Look at how the Weeks class (Cweeks) contains a collection of individual Week classes (Cweek), each Week contains a task collection (Ctasks) and each tasks collection contains a group of individual task objects (Ctask). Using this hierarchy of containers made it very easy to apply the iterator design pattern to traverse the collections. The iterator design pattern “provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation” (Gamma, 1995, p. 257).

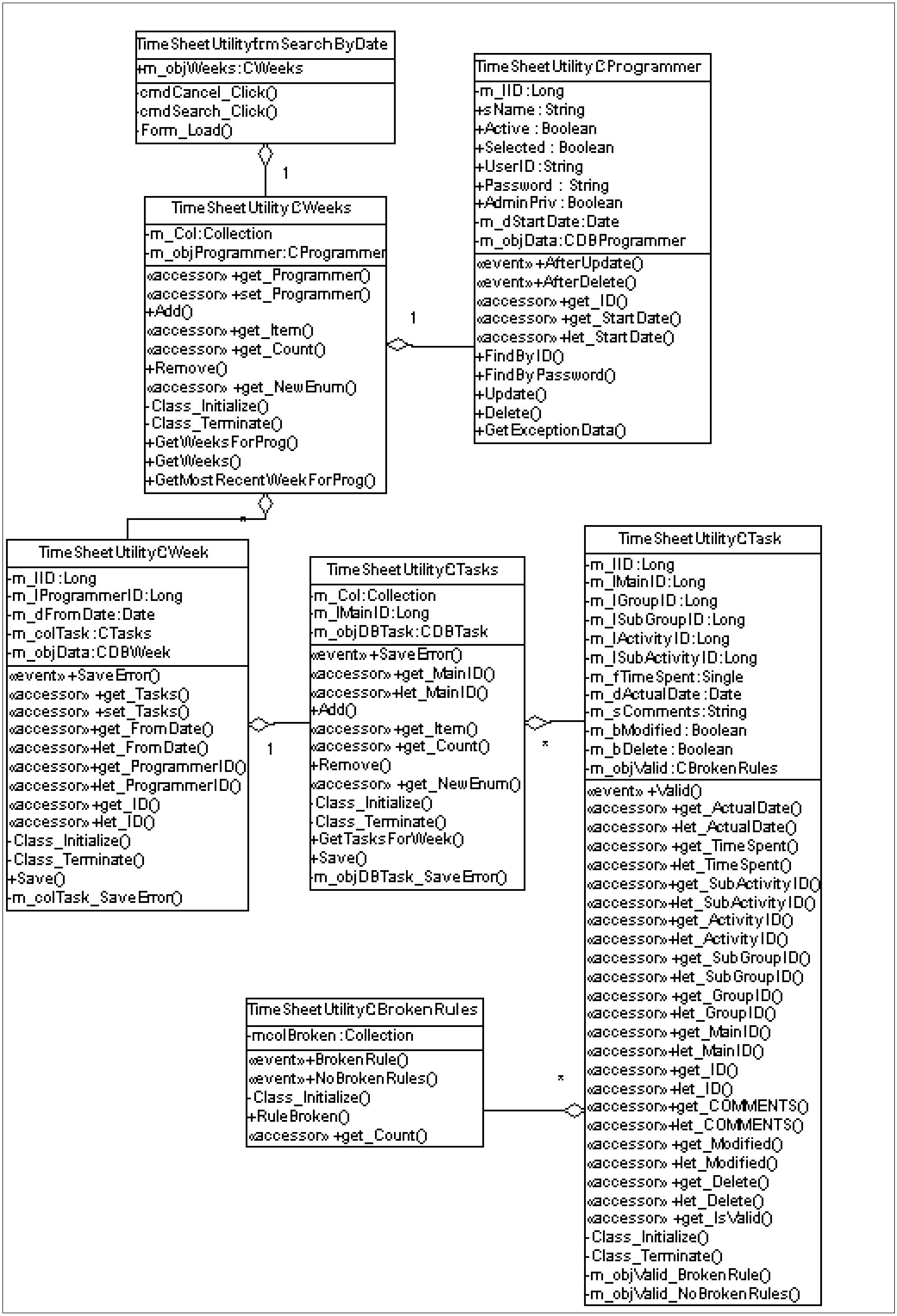


Figure 19 - Form Search by Date (Class Diagram)

Database

The design philosophy of the database was driven by lessons learned from design mistakes made during the development of other applications at Diagnostic Product Corporation (DPC). Databases created for other applications lacked normalization. Many existing databases were structured more like an Excel file than a database entity. There were repeating groups, redundant data, and no relationships established between entities. A primary goal of this project was to successfully apply the rules of normalization to avoid data anomalies that were prevalent in prior applications.

Figure 20 is the entity relationship diagram for the Time Sheet Utility database. It exhibits the application of normalized data. The database consists of 10 tables. The primary purpose of a table is categorized as either lookup, associative, information, or data.

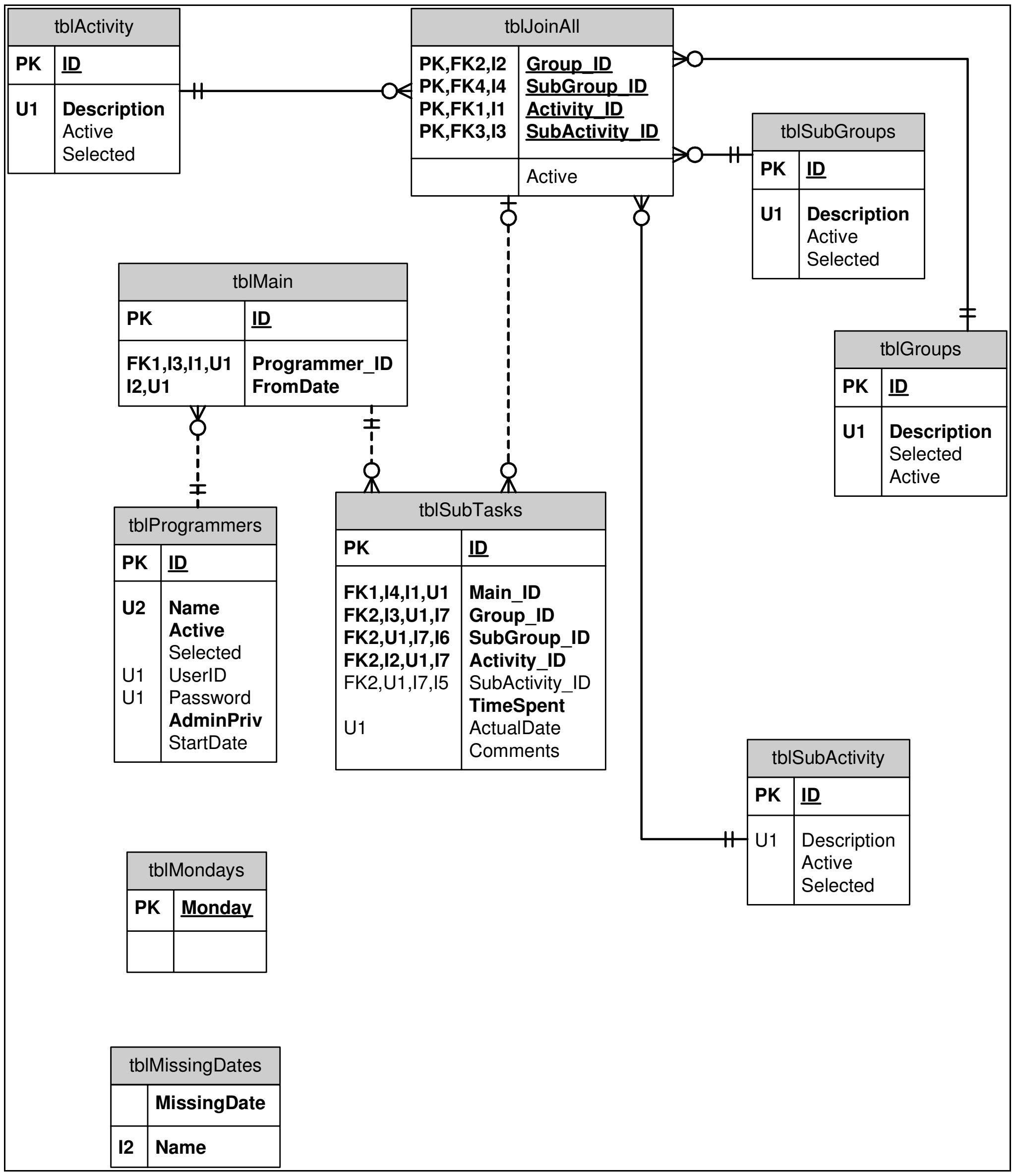


Figure 20 – Entity Relationship Diagram

The purpose of a lookup table is to provide a list of items. The design benefits of using lookup tables are they avoid data redundancy by enforcing a relationship between the lookup table and another entity. For example, tblGroup contains an entry with a description of *Project* whose ID equal 1. The text *Project* is only stored in the lookup table. Any table that needs the group type of *Project* refers back to tblGroup via the ID value of 1. This is beneficial because if the description of *Project* needs to change to something else like *Application* only one record in tblGroup needs to be updated.

The purpose of an associative entity is representing an n-nary relationship. This is a relationship based three or more tables where n represents the number of tables involved in the relationship. For example, tblJoinAll represents an n-nary relationship. It associates all the allowable combination of groups, sub-groups, activities, and sub-activities. This design was chosen because it benefits the application by providing a mechanism to enforce the business rule that a user can only choose valid combinations to record there time against. For example, if a user chooses to record time against the group titled *projects* he should not see *vacation* as a subgroup choice since *vacation* is not a type of project it belongs to another group titled *out of office*. The tblJoinAll table ensures that when the user chooses a group to record time against they can only choose an appropriate sub-group based on that group. The same concept applies as the user traverses down the hierarchy to choose an activity and finally a sub-activities.

Information tables are used to store application data that does not relate to other tables, but are needed to provide data for the application to function properly. For example, tblMissingWeeks is a scratch pad area used by the application as an input into generating the report showing which programmer have not entered their information for given weeks. It is repopulated each time the user attempts to run the report. Table 2 summarizes the descriptions and types of the various tables.

Table 2 – Tables in Database

Name	Type	Description
tblGroups	Lookup	List of groups
tblSubGroups	Lookup	List of sub-groups
tblActivity	Lookup	List of activities
tblSubActivity	Lookup	List of sub-activities
tblJoinAll	Associative	List of valid combinations of groups, subgroups, activities, and sub-activities
tblMain	Data	Each record represents a week for a programmer
tblProgrammers	Lookup	List of Programmers
tblSubTasks	Data	Each record represents a task related to a specific week/programmer via its relationship with tblMain.
tblMondays	Informational	List of Mondays used to validate choices for the starting date of week records
tblMissingDates	Informational	List of missing weeks for programmers

Reports

Most applications have to produce output. It is not enough to show information on the user interface and not provide a mechanism for the user to output the information. The Time Sheet Utility has requirements related to the generation of reports. For the initial release, the application had to produce five reports.

The 1st report was called the *Task Listing Report*. It supplies a list of all activities defined in the system. This report is to help the developers know what categories were available in the system. The 2nd report is titled the

Activity Report and it contained the tasks and the time spent on them for a given time period. This report is used to produce output containing the time spent on various activities. The 3rd report is a *Missing Weeks Report*. It is used by the developer and management to quickly see what weeks have no time recorded against them. It tells who is missing data and for what week(s). Administrative level users use the remaining two reports. They deal with exceptions regarding either a week with over 60 or less than 40 total hours or any single day containing over 12 hours of activities.

Producing the reports was a technical challenge because Management did not provide the developer with a specific report-writing package. In addition, the project was under a constraint not to spend money on any additional tools. Researching and examining the available tools the author was able to utilize object linking and embedding (OLE) automation from the Time Sheet Utility to Microsoft Word. This enabled the application to meet and exceed the reporting requirements since it became easy not only to create complex reports but they could easily be exported to various formats such as Excel, rich-text, or HTML.

What Went Right and Wrong

The design of the application and supporting database afforded many opportunities to apply the research discussed in Chapter Two. This included the chance to apply object-oriented concepts and database normalization.

Some aspects of the project went well while others areas of the project fell short of initial expectations.

The design of the application layer followed the tenets of encapsulation and cohesion. It was helpful to spend the time necessary to design the system prior to implementing it. The use of visual design tools, such as the class diagram, successfully allowed for the logical organization of responsibilities, required from the application, across the classes comprising the application. As the developer, the design worked well when it came to avoiding the pitfall of repeating logic. For example, having a single object responsible for connecting to the database made it very easy to switch the connection properties when the application was deployed. Also, designing each class in the system prior to coding led to the successful structuring of the objects so they encapsulated their state and behaviors.

The design of the database layer required applying many of the theoretical concepts discussed in chapter two. This included defining the entities, enforcing referential integrity and normalizing to 3rd normal form. The normalization process successfully reduced redundancy in the data store by using referential integrity between entities versus repeating data. For example, the programmer table contains the programmer's name. When someone got married, it was only necessary to update a single row in the programmer table rather that all the rows for each week that were for that programmer. The use of relationships between the various entities also reduced data anomalies by ensuring there were no orphaned records between parent and child tables.

Another successful area of the project was the acceptance of the application by the end users. The end users quickly adopted the application. It is believed this was accomplished as a direct result of the high degree of end user involvement throughout the project life cycle. Users' were part of the process from requirements until deployment. Prototypical screens were also created to ensure the concepts assumed by the developer were accurate. Presenting these screen designs to the end users facilitated successful design meetings.

Every project offers opportunities for reflection and self-improvement. This project is no exception. In hindsight, the development tools used for this application were not the best. Visual Basic 6 lacks implementation inheritance. The lack of implementation inheritance resulted in an over reliance on delegation since code could not be reused via implementation inheritance. Microsoft Access lacks many of the features of a true relational database management system (RDBMS). Access is a file server system not a RDBMS and as such it lacks transactional support and transactional recovery. Because of this, Microsoft Access does not handle a large, over 5 concurrent users, environment effectively.

It is true the author was under the constraint not to spend additional money to acquire new development tools. However, it was mistaken to assume this constraint limited the developer to using only Visual Basic 6 and

Microsoft Access since Management provided them. There are free development tools, which support the object-oriented features missing in Visual Basic 6. The same is true for the relational database management system features missing in Microsoft Access. The application layer could have been developed in Java or VB.NET. Sun distributes a free integrated development environment (IDE) for Java called NetBeans. Microsoft distributes the .NET framework for free and even provides a scaled down version of its IDE call Visual Basic Express at no cost. There were also more powerful databases that would not have increased costs. Microsoft Data Engine (MSDE) is full-fledged relational database management system. MSDE has the same application-programming interface (API) as Microsoft SQL Server. MSDE does have some limitation. It limits the maximum size of the database and the number of concurrent connections, but neither of these limitations would have caused it to not meet the needs of this application.

Requirement gathering could have also gone better. Sometimes important stakeholders were not available for meetings and the lack of their input led to additional work later when their needs were finally revealed. It would have been better to reschedule the requirements meeting rather than forging ahead with these people. Also, a consistent group of people to represent the user would have helped as well. Changing the people who represented the user-decreased efficiency since the new personnel had to be brought up to date each meeting.

Overall, design of the application went well. However, the dynamic behavior of the system could have been better defined by the use of sequence diagram. Class diagrams are great for showing static behavior and the relationship between the classes. But, class diagrams do not indicate the sequence of messages between objects. This is better represented with sequence diagrams. The lack of designing the dynamic behavior of the system led to the need to refractor code during the implementation phase. Also, the design could have been better by avoiding unnecessary coupling between the classes in the system. This could have been avoided by passing the object's data as parameters rather than passing the object itself. It was simple to code by passing the entire object, but this did increase coupling.

Summary

The analysis and design of the Time Sheet Utility provided many opportunities to apply the best practices discovered during the research in Chapter two. Requirements were gathered during meetings with the various stakeholders in the project. This included both management and staff personnel.

The design of the graphical user interface and database components caused a smooth transition to implementation since these areas were well thought out prior to coding. The creation of a technical specification that contained class diagrams, an entity relationship diagram, and pseudo-code for every method of each class gave the developer a clear path forward when it came time to implement the application. The presentation of prototype screen designs increased the chance of user acceptance of the application since they were involved in determining how the final product would work.

The database design achieved a degree of normalization consistence with the research on how to design a database that was discussed during Chapter two. Each table contains only attributes related to that entity and heavy use was made of referential integrity between entities to avoid redundant data. The end result was a database design that could meet the demands of outputting and storing data efficiently.

Many aspects of this project went right and some areas could have gone better.

Areas that went well

- Achieved object-oriented design
- Achieved 3rd normal form
- Continuous input from stakeholders/end users
- Achieve high cohesion
- GUI design well liked by end users

Areas that could have gone better

- Unable to leverage implementation inheritance due to VB6 limitations
- Some requirement were vague
- Design dynamic behavior with sequence diagrams
- Decrease coupling of objects between architectural layers.

In summary, the analysis and design of the Time Sheet Application went very well. This conclusion can best be supported by how effectively the implementation and coding phases of the software develop went. In addition, the acceptance and current use of the application by the software department supports this conclusion.

Chapter Four - Project History

How the project began

The Time Sheet Utility project was initiated to enable the management at Diagnostic Product Corporation to better understand how developers' work time was being allocated across various projects and activities. The need for this utility was exacerbated by the fact that current time reporting was being done in an unorganized ad-hoc manner that failed to facilitate determining where the department's resources were spending their time.

The initiator of the project was the Director of the Software Department. He asked the Manager of the Software Department to come up with a solution for tracking resource allocation more efficiently. The Manager assigned the author of this paper with the task of analyzing, designing, implementing and testing a solution.

The first project related activity was a meeting with the key stakeholders to discuss what functionality was desired. The attendees consisted of the Director, Manager, a senior programmer and the Project Manager. The goal of this initial meeting was to define the scope and feasibility of the project.

How the project was managed

The project was managed following the principles of iterative development. It was decided to break the development of the project into iterations. In iterative development, successive iterations add more functionality to the application. It was decided the focus of the first iteration would be to implement the necessary functionality to allow for the collection of data, maintenance of system related tables, and limited reporting capabilities.

The project team consisted of a single person who performs the functions of project manager, system analyst, developer, and tester. The small makeup of the project team impacted how the project was managed. A small team requires less meetings and discussions since there is no interaction between the various roles previously mentioned. Due to this single member project team environment, many aspects of the software development lifecycle could be accelerated. For example, a code review could be conducted immediately since there was no need to coordinate a meeting with a team of reviewers.

Project management best practices were followed when it came to interaction with stakeholders. A key objective to making this project successful was acceptance of the final product by the developers and management. To help ensure this would happen, the users was continually involved in the development process to ensure the project was meeting their needs. For example, users were shown screen mockups prior to implementing the real screens. Continually involving the stakeholders was an important aspect of managing the project. Another important tool in managing the project was the project plan. This document helped to define important information including who are the stakeholders, scope, constraints, and schedule.

Significant Events & Milestones

Table 3 lists the significant events and milestones of the project.

Table 3 – Significant Milestones and Work Breakdown Structure

TASK	DURATION (IN DAYS)	MILESTONES
1. Requirement gathering (analysis)	**11**	**8/29/2005**
1.1. Interview stakeholders	5	
1.2. Perform analysis	5	
1.3. Write detailed requirements	0.5	
1.4. Review requirements with stakeholders	0.5	
2. Planning	**1**	**8/31/2005**
2.1. Estimate time and scope	0.25	
2.2. Establish stakeholders	0.25	
2.3. Determine constraints	0.25	
2.4. Write project plan	0.25	
3. Design	**26**	**10/6/2005**
3.1. Design user interface	8	
3.2. Write user specifications	2	
3.3. Design database	5	
3.4. Create entity relationship diagram	0.5	
3.5. Review user specifications with stakeholders	0.5	
3.6. Write technical specification	8	
3.7. Do personal review of technical specifications	2	
4. Implementation	**20**	**11/4/2005**
4.1. Write code	15	
4.2. Do personal review of code	2	
4.3. Create database	2	
4.4. Do personal review of database	0.5	
4.5. Check code and database under version control	0.5	
5. Testing	**4**	**11/11/2005**
5.1. Write test plan (2 days)	2	
5.2. Run unit test plan (2 days)	2	
6. Deployment	**2**	**11/15/2005**
TOTAL DURATION	**62**	

Changes to the plan

The plan had minimal changes during the life of this project. Total duration was increased due to the addition of a deployment phase. The original scope of the project was through unit testing and did not include the deployment of the application to the end user. This addition increased the duration of the project by two days.

Another change to the original plan was the removal of the creation of a hazard analysis. The hazard analysis task was not deemed necessary since this application was not to be used in conjunction with a medical device. Diagnostic Product Corporation’s core business is the development of medical diagnostic instruments. Normally, it is required to perform a hazard analysis as part of the software development lifecycle. However, since this application was an internal utility not to be used by external clients, the Director of the Software Department authorized a deviation from this software development lifecycle requirement.

Goal Evaluation

The Time Sheet Utility was developed with specific goals and objectives. This section shall summarize these goals and objectives and how successful the application was a meeting them.

The application utilizes a networked database that allows multiple user access to needed data. In addition, it produces reports that can be used by both developers and management to interpret time spent. The Time Sheet Utility provides management the ability to better assess the needs of the department and the various projects within the department. It accomplishes this by providing immediate and accurate data necessary for management to make educated decisions about the allocation of staff across multiple projects.

Another goal for the project was to complete the project on time and under budget. As far as being completed on time, the project was finished by the date promised in the project plan. In relation to meeting the budgetary goal, it was finished using one human resource and without the purchase of any additional hardware or software.

An additional goal of the project was to have the end-users and stakeholders complete a survey about the acceptance of the application to indicate if requirements were met. Unfortunately, to date this survey has not been given so therefore it is more difficult to assess whether or not this goal has been met. However, the system has performed very well since being deployment. To date only two minor defects have been reported. Management has provided its developers with a standard way to record time spent and can analyze the data from the central data repository, and all developers use this application.

What Went Right and Wrong

During the project history, it is satisfying to report that more aspects of the project went right than wrong. A key decision that resulted in many of the successes on this project was the proper amount of time was spent on analysis and design prior to implementation. In the past, the developer has often rushed to coding without spending enough time collecting requirements and designing a solution that meets those requirements. On this application, much more time was spent designing and analyzing. This upfront work led to a very smooth transition to the implementation and testing phases of the project. It is believed the creation of prototype screens prior to coding the application led to the high-level of user acceptance. The author successfully gathered requirements from both management and staff to create an application that met the needs of both administrative and staff users.

The work breakdown structure created for this project supported the successful creation of an object-oriented system and relational database. It did this by allowing proper time for the necessary tasks to do this type of development. The project met budgetary and schedule commitments as outlined in the project plan during the inception of the project. The final testament to how well the project went is the application is currently used across the entire department.

Some aspects of the project could have gone better. In an effort to meet schedule commitments time was cut from the testing phase of the project. There should have been more time allocated for testing prior to deployment. However, to date only two defects have been found by the users. Neither of which has resulted in any inaccurate data being stored. Also, more time should have been spent researching the development tools used on the project. In hindsight, it would have been preferable to use an alternative database system to Microsoft Access. The same could be said for the choice to use Visual Basic 6. There should have been more exploration into other programming languages prior to choosing Visual Basic 6.

Project Variables and Their Impact

The software development process has many elements that need to come together to reach the desired result. The project plan and work breakdown structure provides a map towards the intended objectives. However, it is to be expected that not everything will go according to plan. A wise person once stated the only constant is change. It is important to realize what the variables are on a particular project and how they may impact reaching the intended outcome. Understanding project variables and preparing for their impact is the basis for risk management.

Early in the project, one variable was the requirements gathering process. Each week a meeting was held to discuss what functionality was needed from the Time Sheet Utility. The reason requirement gathering was seen as a variable was because each week different stakeholders showed up to the meeting. The variability in the attendance at these meeting impacted the project because each week different people stated different objectives and requirements. The reaction to this variability was the creation of a consistent group of attendees for these requirement meetings. Once this was done, more productive meetings were held since we did not spend the first half of each meeting trying to fill in new attendees on what they missed since the last gathering.

Another variable on the project was resource allocation. The project plan called for the developer of this application to devote 90% of his time to the project. However, four weeks into the project the Technical Lead for the software department reassigned the developer to another tasks. The impact on losing this resource would be the project end date would not be met. The reaction to the potential lose in resources was to negotiate the situation with the project sponsor, the Director of Software Department. After discussing the situation, it was decided not to pull the resource from this project but to provide a different resource to the Technical Lead. The potential impact of this variable was negated by negotiations.

Testing and Implementation

A total of four days was allocated towards developing and executing the testing phase of the project. The types of testing included both unit (white box) and integration (black box) testing. The purpose of unit testing is to verify the proper internal working of a specific component. The goal of this level of testing was to ensure all branches of code within each method work as designed. Integration testing was performed after unit testing was completed. The goal of integration or black box testing was to verify the various software components worked well together. It validates the interfaces and interactions between the components that make up the application.

One issue with the testing phase was not enough time was allocated in the project plan to complete 100% code coverage in unit test. A discussion was held between the project sponsor and the project manager to discuss the situation. It was decided not to add more time to the project schedule to accomplish 100% code coverage in unit test. Highly improbable paths would not be tested since the return on investment for the time invested to accomplish this was not justified in the mind of the project sponsor. An example of the type of code path not tested was when a local variable might have lost scope unexpectedly.

In hindsight, the decision not to add more time to the project schedule to accommodate addition testing appears to have been the right decision. This conclusion is based upon the fact that only two defects have been found since the application was deployed. The application consisted of over 4,000 lines of code. The defect density in system test was less than one defect per one thousand lines of code (KLOC). It is believed the reason for the low defect density was quality was built and not tested into the application. To paraphrase, quality is achieved by proper analysis and design not through testing.

Twenty days were allocated to the implementation phase of the project. Seventeen days were allocated to the implementation of the application code and three days were allocated for database implementation. The application code was completed in fifteen days, but the database code took six days. The overall time it took to complete implementation was twenty-one days. Implementation went well overall.

The report generation process was difficult since there was a lack of financial resources to buy a report-writing component such as Crystal Reports. It was not until the developer discovered the ability to leverage automation to Microsoft Word that a path forward became possible. However, using automation created new issues. The automation call would open up a instances of Microsoft Access loaded with the database. This gave the application user access to the entire database! After an extensive research into the Windows application-programming interface (API), a function call was found that would hide the Microsoft Access window from the user.

Findings & analysis results

The Time Sheet Utility met schedule expectations by being completed within sixty-two days from project initiation. The project met budgetary goals by being completed using a single resource and leveraging development tools currently available to the software department at DPC. The project met quality goals by successfully being adopted for use by the entire software department and having only two defects found in system test. Neither of these defects was major enough to hamper usage by the developers. In addition, the system was deemed fit for use by the Management. It successfully met management's goals of centralizing data and standardizing the data collection protocol.

The application and associated database did meet the requirement to support a multiple user environment. The application is most heavily used on

Fridays. This is when timesheets are due. It has been observed that eight developers were using the application at the same time without any degradation in the performance of the system. Once a developer lost data entered into the system when the company's network went down while he was entering data. This data could not be recovered since the backend; Microsoft Access does not maintain a transaction log. It was necessary for the reentry of the missing information. This incident led to the conclusion Microsoft Access is not the best backend for disaster recovery.

Summary

Project management is a complex discipline. Being a project manager requires a diverse set of skills. These skills related to various areas including managing of scope, time, cost, and risk and communication process areas. The success of this software engineering project began with proper project initiation that led to clearly defining out the scope of the project. Equally important to the success of this or any other project is developing a well-defined plan. The work breakdown structure was important in laying out the path forward for all involved. However, having a good plan is only one step toward success; you must then execute the plan and measure your progress against it.

Analysis and design must receive the proper amount of attention when developing an application such as the Time Sheet Utility. This was a key success factor in this project. Not giving enough attention to these phases of the software development cycle often leads to a implementation that is not well thought out. It would be analogous to building a house without a blueprint. It is equally important to not circumvent proper testing of the system prior to deploying it. User acceptance is very important to success and there is no faster way to lose this than releasing an application that is defective and not fit for use.

As mention previously, well-defined scope and clear requirements are key factors towards the project's success. Not having consistent representation from the users' was detrimental to the effective gathering of requirements. In hindsight, a dedicated team of users' should be made available to facilitate the effective analysis of the needs of the system. Having clear and timely communication with the project stakeholders is also important. When schedule or resource issues arouse on the project, the effect of these project changes were mitigated by effective and timely communication that facilitated effective negotiation.

Overall the project met expectations, yet as indicated in this chapter, there was many lessons learned that could be applied on future development endeavors. The author looks forward to these future opportunities to better refine his project management and software development skills.

Chapter Five - Lessons Learned

What was learned from the project experience

This project represented a significant effort to apply the concepts and principles of object-oriented analysis and design. The system used decomposition effectively by looking at how to assign roles and responsibilities across the various objects that made up the application domain. As developer, it was a learning experience to see how proper use of delegation and encapsulation creates a robust and maintainable application.

This project afforded a unique opportunity to apply many of the tools and techniques learned as part of the curriculum at Regis University. Some of the tools and techniques applied included various Unified Modeling Language diagrams and database design. The application of class and deployment diagrams provided a new way to communicate ideas with peers and stakeholders. It was educational to see the effect of using a common method for communicating design ideas.

The database design afforded an excellent chance to apply the theoretical concepts of Dr. Codd’s normal forms. This was the first normalized database created at DPC. It will provide an example of how to follow the theory of normalization. No longer will database design be driven by the immature process of transferring excel sheet style inputs or reports directly to table structures. Establishing proper entities and their relationships leads to a well-formed database that can efficiently store information while not compromising the integrity of the data it is storing.

The software development lifecycle approach used on this project provided a roadmap on how to logically proceed through the various stages of software development from initial analysis to deployment. The iterative approach to software development succeeded in guiding the project to completion. Each phase established clear deliverables and objectives that taught the importance of coordinating efforts across all project participants. Prior experiences were often a rush to code without putting in the proper amount of analysis and design. This project illustrated how important it is to not short change the amount of effort put into analysis (requirement gathering) and design.

Finally, many tradeoffs must be made as part of managing a project. Scope, time and cost are all competing aspects of project management. If one of these “triple threats” of project management changes the other two are affected. For example, adding a new requirement will mean taking more time to complete the project based on current resources. However, adding additional resources will hold the finish date the same, but now cost will increase due to the added labor cost. The key lesson here is all issues can be resolved so long as they are discussed. The art of negotiation was pivotal in managing the project because when options were discussed tradeoffs could be sensibly explored.

What would have done differently

While overall the project went very well, there are definitely some aspects of the project that could have been done differently. Some areas the system architect would have done differently included: using a fully object-oriented language, more robust backend, and allowing for automatic or no-touch deployment.

Use fully object-oriented language like Visual Basic .NET

The lack of implementation inheritance led to increased effort and code since it was necessary to resort to delegation instead. The data access layer had more redundancy in the code due to the need to repeat certain functionality in each of the data access classes. Implementation inheritance would have been most useful in situations like this. Visual Basic .NET uses a disconnected data access model, which would have created a more scaleable system by limiting the number of locks on the data in a multi-user environment.

Use Microsoft Data Engine vs. Microsoft Access (Jet Engine) for the backend.

Access is fine for very small workgroups, but it really was never meant to be a high-powered backend system. It is uses a file system concept, which lacks functionality afforded by a true relational database management system like the Microsoft Data Engine (MSDE). One of the features not supported with the file system approach consist of not supporting transactions effectively since there is no transaction log. The transaction log keeps track of pending changes to the data. If Microsoft Access crashes you lose any pending changes. Also, Access does not support connection pooling.

MSDE has some limitation but basically affords most of the same functionality as Microsoft SQL Server 2000. According to the Textratex website,

> MSDE 2000 is a redistributable version of SQL Server 2000. It is a database engine provided by Microsoft that is based on the core SQL Server technology and supports single- and dual-processor desktop computers. In other words, MSDE 2000 is a scaled down copy of SQL Server. MSDE 2000 was introduced to provide application developers a database engine that is more powerful than the Jet engine and at the same time expandable to SQL Server. It is ideal for client applications requiring an embedded database and websites serving up to 25 concurrent users (2006).

Some of the limitations of MSDE include it limits the number of active connections to five then performance degrades, the size of the database cannot exceed two gigabytes, and it does not include any client tools for development. Each of these limitations would not have adversely affected the ability to MSDE as the backend on this project. Additionally, MSDE is free so it would not have increased the cost of the project.

<u>Consider architecting as a smart client application.</u>

One limitation that the current architecture has is it requires the user to always be connected to the backend in order to enter data. This means the user cannot enter data when working from home. They must wait until they come back to work to record time spent. The Smart client application architecture could support both a connected/disconnected model so that users could still enter data while disconnected and the upload that data when they return to the office (Hollis, 2004).

A smart client could also better handle deployment issues. Currently, every upgrade now requires a manual installation at each user's computer. Smart client architecture could support automatic deployment by having the application determine if it is the latest version and if not trigger the upgrade process automatically (Hollis, 2004).

Did the project meet initial expectations?

The expectations of the Time Sheet Utility Project was to provide a standardized mechanism for collecting the data reflecting how much time developers were spending on various activities. The goal of this utility was to automate the collection of data and provide for uniformity of data collected. This utility established this standardization by collecting data in such a way that business rules were properly enforced. The expectation was this data would provide Management with timely and accurate information to make better decisions.

The software has been in use for over six months. Management and the developers have been very satisfied with this application. It has succeeded in meeting everyone's initial expectations. Management has already utilized the data collected so far to initiate various department wide initiatives to increase the productivity of the department. For example, once Management realized how much time was spent by developers on supporting technical services it was decided to form a core team whose members would have the responsibility of handling all inquiries from the technical services department. In addition, the data collected has been used to support the justification to upper management for adding software developers to the department. Another example of how the data collected by the application has been used was the reallocation of current resources to different a project since management decided the time being spent on the rewrite was not justified.

Currently, all software developers and the quality assurance department personnel use this utility to record their time. The successful deployment of the application has led to the standardization of collecting resource allocation data. No longer is it required to manually collate a variety of ad-hoc reports to determine aggregate totals with a click of a button the data is immediately available, which used to take days. The application has successfully met expectations surrounding the enforcement of business rules. Users and management are notified when data is not recorded in a expeditious manner. The choice of tasks is predefined and standardized yet easily extendable. The selection of tasks is limited by the application. An administrative user can add additional tasks as necessary. Overall, it is the author's judgment based on the reaction of management and staff that the initial expectations of the project have been met and exceeded.

What would be the next stage of evolution for the project?

The Time Sheet Utility has been successfully deployed. The initial set of features is done. However, there are still areas where the project could evolve. Going forward the next stage of development might involve the following enhancements:

- Upgrade the backend to MSDE.
- Allow for offline data entry (smart client architecture).
- Upgrade the code to Visual Basic .NET from Visual Basic 6
- Add additional reporting capabilities to provide deeper analysis of the data collected.

Many of the ideas for the next iteration of the project were extrapolated from the areas the author might have done differently. Other ideas came from new needs of the end users.

The current backend could be more robust. As discussed earlier, Microsoft Access is not a fully featured relational database management system. It would be beneficial to upgrade the database to the Microsoft Data Engine (MSDE). This modification would improve multi-user support and data access reliability. The application would reap the benefits of better code reuse and being more maintainable if the code base was rewritten using a fully object-oriented language such as Visual Basic .NET. This would also allow the architecture to be modified to support a disconnect data access model. The .NET framework inherently supports disconnected data.

Currently, the application produces a limited number of administrative and staff reports. Since being released, Management has requested additional reports should be generated by the application. Management could definitely benefit from additional reports. These reports would present various views of the information contained in the database. For example, the Director of Software requested a report showing the percentage of time spent on each project and each subtask within those projects.

Another possible evolution of the current application could be supporting the ability to enter data while not connected to the network. Right now It is required that the user be connected to the network database when they enter their information. This is limiting since some developers work from home and the network connection is not available. These users would benefit if they could enter data locally and then periodically upload their data to the network database. Microsoft has been espousing the adoption of *Smart Client Architectures* to support just this functionality. The disconnected data access model supported by ADO.NET could be leveraged to make this feature available in the future. Also, the *Smart Client Architecture* would facilitate easier deployment of the application in the future since newer versions of the application could be automatically downloaded to the client for a web server (Hollis, 2004).

Conclusions/recommendations

The goals of the project were met. The application is currently being used throughout the software department. Management is using the data as a basis of making resource allocation related decisions. Object-oriented design can be successfully applied to create a cohesive, maintainable architecture. Putting the proper ratio of project time into analysis and design versus implementation was a key factor in contributing to successful performance of the application. Normalization is the correct approach to database design. Successful application development starts with getting the correct requirements and continuing user involvement throughout the software development process.

Some recommendations for this project are to begin planning a second iteration of project development to create additional management reports and allow for disconnected data access. It is also recommended that future projects assign a dedicated group of end users who would consistently attend user group meetings to help define the systems appearance and functionality. Having a dedicated group would be beneficial because it would reduce wasted effort on having to accommodate explaining the prior project status to new attendees at every meeting.

Another recommendation is to conduct peer reviews of project deliverables. Only one resource was allocated to this project. This meant the same person who created the deliverables (technical specifications and source code) performed the review of those deliverables. It is recommended to allocate at least some time for an additional resource to perform peer reviews of design and code artifacts. This would help in two ways. It should reduce defects by providing additional input from another person. It would also help to transfer knowledge of the application from the developer to another person. This transfer of knowledge will help mitigate the risk of the sole developer leaving the company and yet having no one else who understands the inner workings of the project.

Summary

Many lessons were learned during the development of this application. This project has afforded an excellent opportunity to apply many of the theoretical concepts discussed throughout this paper. The application layer provided the opportunity to use object-oriented concepts. The database layer required the application of relational database design using normalization.

There were definitely some areas that could have been done differently based on the experiences gained implementing this software solution. Some areas the system architect would have done differently included: using a fully object-oriented language, more robust backend, and allowing for automatic or no-touch deployment. It is believed these architectural changes would have resulted in a more robust application.

Overall, the application did exceed expectations. It has been in production for over six months and continues to be used by staff and management to track resource allocation. The application succeeded in providing a central repository for data, standardizing data collection, and supporting macro level analysis of the collected data.

The first release of the Time Sheet Utility was successful. The next iteration of development could include further enhancements in functionality by providing more comprehensive management reporting and implementing upgrading the backend to the Microsoft Data Engine and the front end to Visual Basic .NET. It is believed these enhancements will increase the scalability of the application to the enterprise level.

In conclusion, it the author’s determination based on research and practical application that object oriented analysis using the iterative development lifecycle lead to the success of the project. It is also believed database design is best accomplished when following the theories established by Dr. Codd for normalization of data.

References

Database Normalization Retrieved January 15, 2006 from http://en.wikipedia.org/wiki/Database_normalization.

MSDE vs. SQL Server Retrieved April 4, 2006 from http://www.teratrax.com/articles/msde_vs_sql_server.html

Fowler, M. (2000). *UML Distilled 2nd Edition – A Brief Guide To The Standard Object Modeling Language*, Reading, MA: Addison-Wesley.

Gamma, E., Helm, R.,Johnson, R, & Vlissides, J.(1995). *Design Patterns Elements of Reusable Object-Oriented Software*, Boston, MA: Addison-Wesley.

Goren, B. (2004). *Normal Forms* Retrieved January 15, 2006 from http://www.trumpetpower.com/Papers/Normal_Forms.

Hollis, B. (2004). *Back To The Future With Smart Clients* Retrieved April 7, 2006 from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnreal/html/realworld03232004.asp

Kroll, P. (2004). *Transitioning from waterfall to iterative development* Retrieved December 28, 2005 from http://www-128.ibm.com/developerworks/rational/library/4243.html.

Newmann, P. (2002). *Commercial Off The Shelf Software: Buyer Beware* Retrieved January 18, 2006 from http://www.srmmagazine.com/issues/2002-03/cots.html.

Puttick, R. & Tkach, D. (1994). *Object Technology In Application Development*, New York, NY: The Benjamin Cummins Publishing Company Inc.

Roman, S. (1997). *Access Database Design & Programming*, Sebastopol, CA: O'Reilly & Associates Inc.

Rubin, P. (2001). *Build or Buy* Retrieved February 18, 2006 from http://paul.rubinsoftware.com/Musings/BuildOrBuy.html.

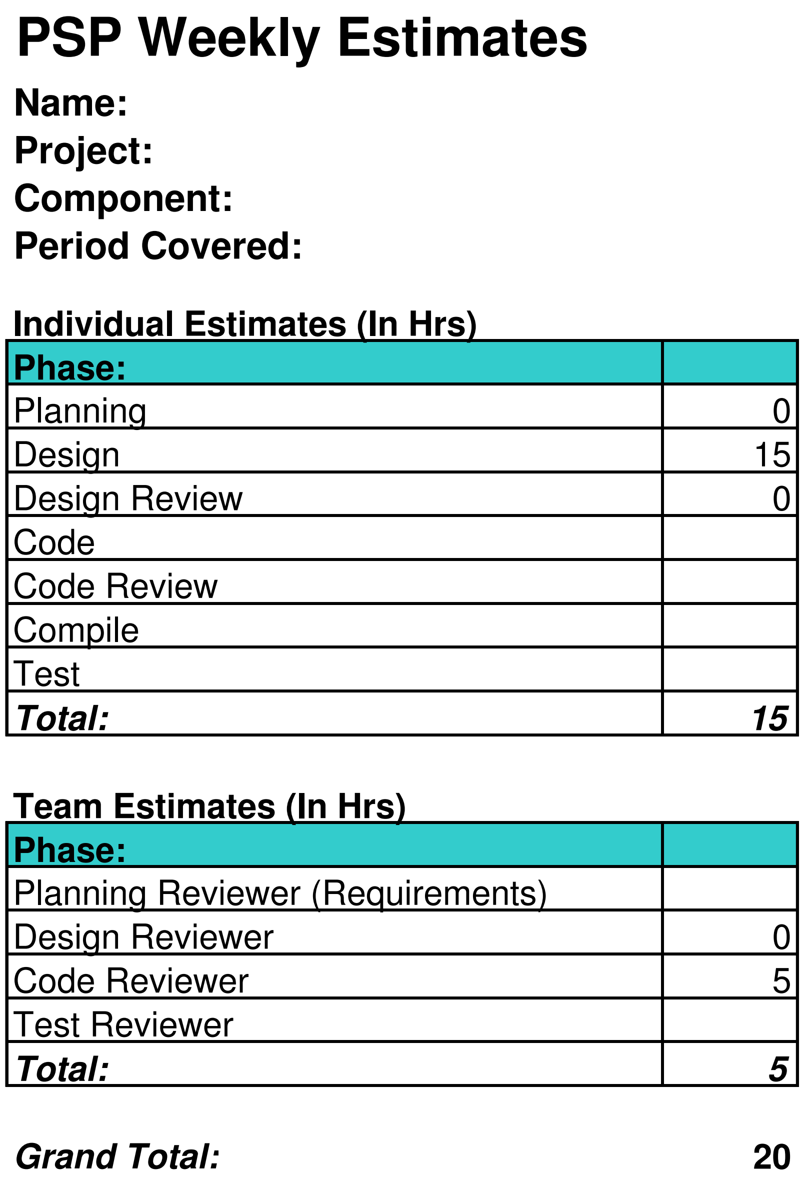
Stiefel, M. (2005). *Thoughts on Software Development* Retrieved February 23, 2006 from http://www.reliablesoftware.com /weblog/2005_10_23_archive.html.

Traylor, P. (2006). *To Build or To Buy IT Solutions* Retrieved February 18, 2006 from http://www.infoworld.com/article/06/02/13/74880_07 FEbuildbuy_1.html.

Walton, B. (2004*). Iterative vs. Waterfall Software Development: Why Don't Companies Get It* Retrieved February 18, 2006 from http://computerworld.com/printthis/2004/0,4814,90325,00.html.

Appendixes

Appendix A - Standardized Excel Time Sheet



PSP Weekly Estimates

Name:
Project:
Component:
Period Covered:

Individual Estimates (In Hrs)

Phase:	
Planning	0
Design	15
Design Review	0
Code	
Code Review	
Compile	
Test	
Total:	***15***

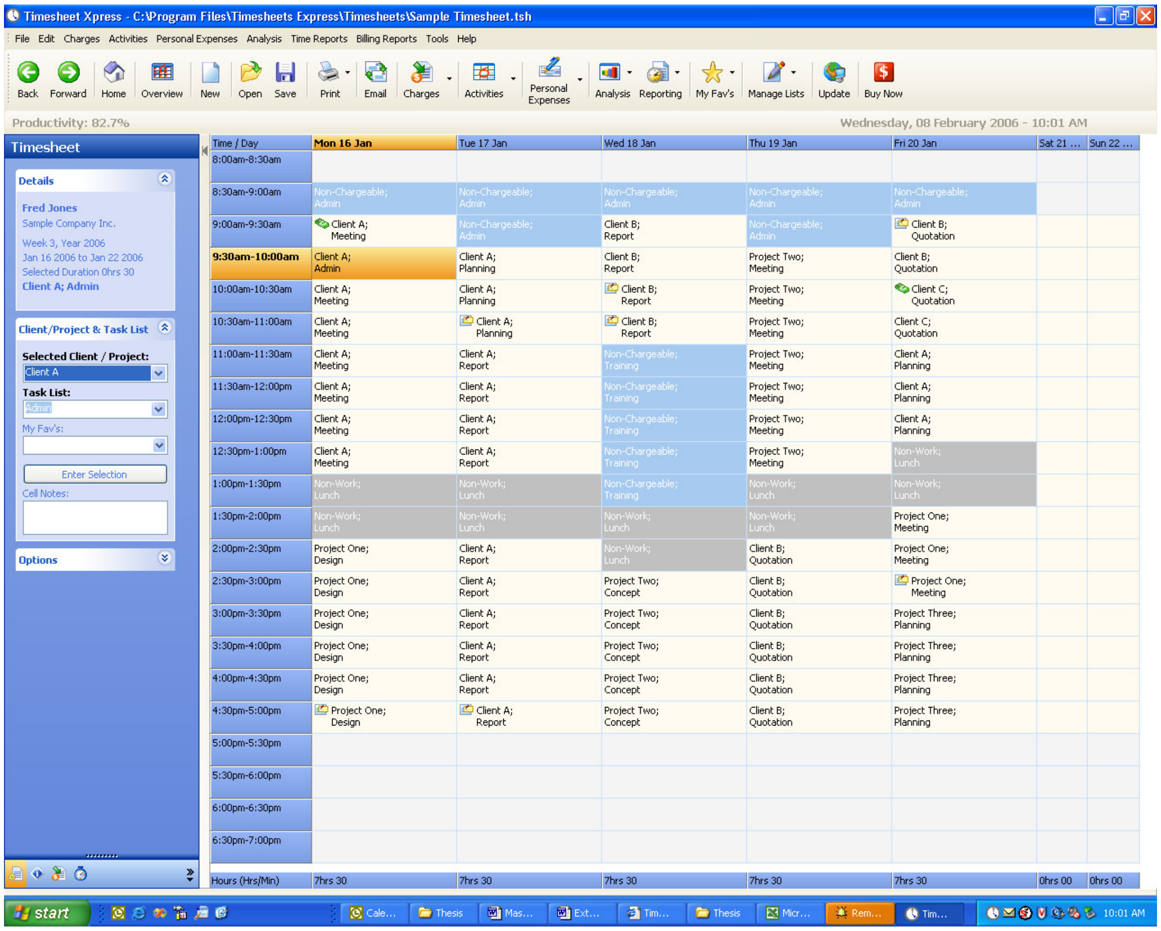
Team Estimates (In Hrs)

Phase:	
Planning Reviewer (Requirements)	
Design Reviewer	0
Code Reviewer	5
Test Reviewer	
Total:	***5***

Grand Total: **20**

Figure A1 – Proposed Alternate Solution (Excel Time Sheet Format)

Appendix B - Time Sheet Express User Interface



Timesheet Xpress - C:\Program Files\Timesheets Express\Timesheets\Sample Timesheet.tsh

Productivity: 82.7%

Wednesday, 08 February 2006 - 10:01 AM

Timesheet

Details

Fred Jones
Sample Company Inc.
Week 3, Year 2006
Jan 16 2006 to Jan 22 2006
Selected Duration 0hrs 30
Client A; Admin

Client/Project & Task List

Selected Client / Project: Client A
Task List: Admin
My Fav's:
Cell Notes:

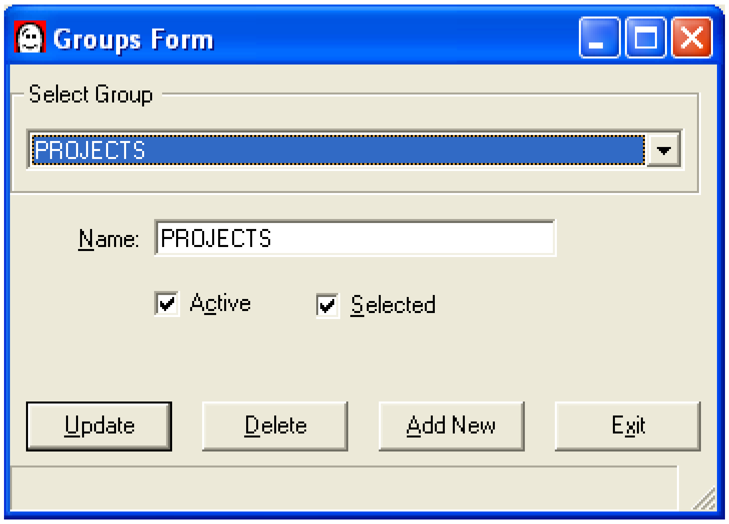
Time / Day	Mon 16 Jan	Tue 17 Jan	Wed 18 Jan	Thu 19 Jan	Fri 20 Jan	Sat 21 …	Sun 22 …
8:00am-8:30am							
8:30am-9:00am	Non-Chargeable; Admin	Non-Chargeable; Admin	Non-Chargeable; Admin	Non-Chargeable; Admin	Non-Chargeable; Admin		
9:00am-9:30am	Client A; Meeting	Non-Chargeable; Admin	Client B; Report	Non-Chargeable; Admin	Client B; Quotation		
9:30am-10:00am	Client A; Admin	Client A; Planning	Client B; Report	Project Two; Meeting	Client B; Quotation		
10:00am-10:30am	Client A; Meeting	Client A; Planning	Client B; Report	Project Two; Meeting	Client C; Quotation		
10:30am-11:00am	Client A; Meeting	Client A; Planning	Client B; Report	Project Two; Meeting	Client C; Quotation		
11:00am-11:30am	Client A; Meeting	Client A; Report	Non-Chargeable; Training	Project Two; Meeting	Client A; Planning		
11:30am-12:00pm	Client A; Meeting	Client A; Report	Non-Chargeable; Training	Project Two; Meeting	Client A; Planning		
12:00pm-12:30pm	Client A; Meeting	Client A; Report	Non-Chargeable; Training	Project Two; Meeting	Client A; Planning		
12:30pm-1:00pm	Client A; Meeting	Client A; Report	Non-Chargeable; Training	Project Two; Meeting	Non-Work; Lunch		
1:00pm-1:30pm	Non-Work; Lunch	Non-Work; Lunch	Non-Chargeable; Training	Non-Work; Lunch	Non-Work; Lunch		
1:30pm-2:00pm	Non-Work; Lunch	Non-Work; Lunch	Non-Work; Lunch	Non-Work; Lunch	Project One; Meeting		
2:00pm-2:30pm	Project One; Design	Client A; Report	Non-Work; Lunch	Client B; Quotation	Project One; Meeting		
2:30pm-3:00pm	Project One; Design	Client A; Report	Project Two; Concept	Client B; Quotation	Project One; Meeting		
3:00pm-3:30pm	Project One; Design	Client A; Report	Project Two; Concept	Client B; Quotation	Project Three; Planning		
3:30pm-4:00pm	Project One; Design	Client A; Report	Project Two; Concept	Client B; Quotation	Project Three; Planning		
4:00pm-4:30pm	Project One; Design	Client A; Report	Project Two; Concept	Client B; Quotation	Project Three; Planning		
4:30pm-5:00pm	Project One; Design	Client A; Report	Project Two; Concept	Client B; Quotation	Project Three; Planning		
5:00pm-5:30pm							
5:30pm-6:00pm							
6:00pm-6:30pm							
6:30pm-7:00pm							
Hours (Hrs/Min)	7hrs 30	7hrs 30	7hrs 30	7hrs 30	7hrs 30	0hrs 00	0hrs 00

Figure B1 – The main screen of the Time Sheet Express user interface

Appendix C - Graphical User Interface (Additional Screenshots)

Table C1 - Menu Structure

Level 1	Level 2	Level 3
File	→Add Current Week	
	→Delete Activity	
	→Exit	
Search	→By Date	
	→By ID Number	
Reports	→Activity Report	
Admin	→E-Mails	
	→Table Updates	→Programmers
		→Groups
		→Sub-Groups
		→Activities
		→Sub-Activities
	→Database Utilities	→Compact Database
		→Repair Database
Help	→About	



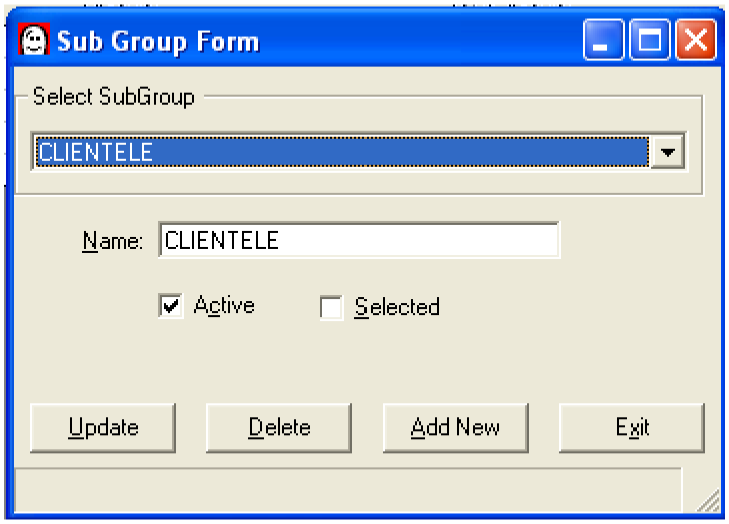
Select Group

PROJECTS

Name: PROJECTS

☑ Active ☑ Selected

Figure C1 - Group Screen



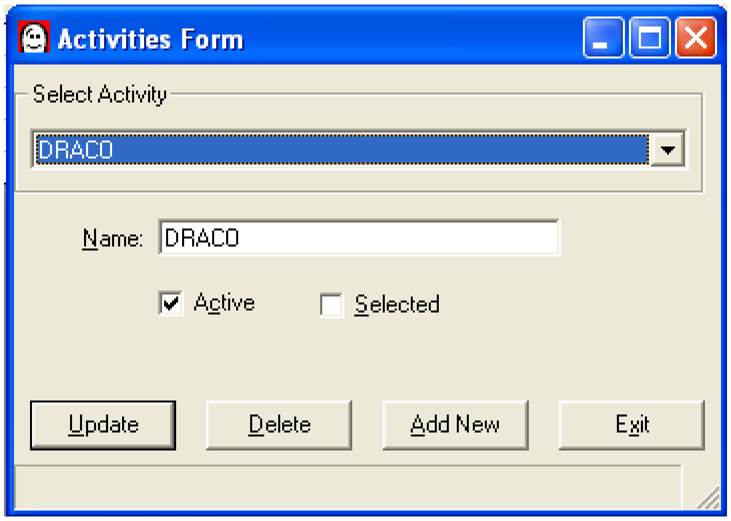
Select SubGroup

CLIENTELE

Name: CLIENTELE

☑ Active ☐ Selected

Figure C2 - Sub Group Screen

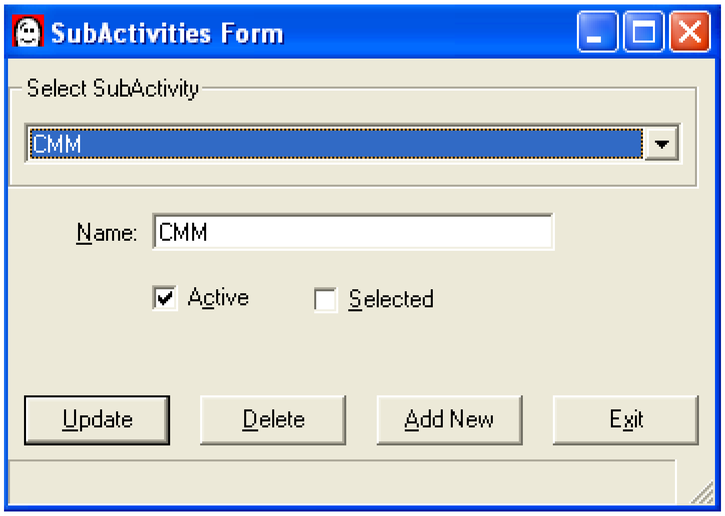

Activities Form

Select Activity: DRACO

Name: DRACO

[✓] Active [ ] Selected

Figure C3 - Activities Screen

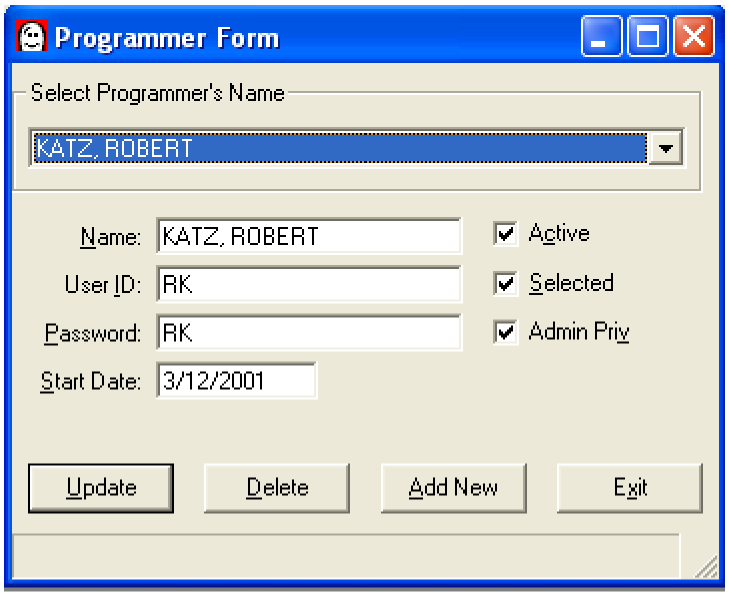

SubActivities Form

Select SubActivity: CMM

Name: CMM

[✓] Active [ ] Selected

Figure C4 - Sub Activities Screen



Programmer Form

Select Programmer's Name: KATZ, ROBERT

Name:	KATZ, ROBERT
User ID:	RK
Password:	RK
Start Date:	3/12/2001

[✓] Active
[✓] Selected
[✓] Admin Priv

Figure C5 - Programmer Screen