SOFTWARE DEVELOPMENT PROCESS: WEB-BASED PAVEMENT MANAGEMENT SYSTEM AS CASE STUDY

BY

WARIT DURONGDEJ

BACHELOR OF ENGINEERING
CHULALONGKORN UNIVERSITY, 1999

SUBMITTED TO THE DEPARTMENT OF CIVIL AND ENVIRONMENT ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ENGINEERING IN CIVIL AND ENVIRONMENT ENGINEERING

AT THE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2001

AUTHOR .....
DEPARTMENT OF CIVIL AND ENVIRONMENTAL ENGINEERING
MAY 11, 2001

CERTIFIED BY .....
GEORGE KOCUR
SENIOR LECTURER OF CIVIL AND ENVIRONMENTAL ENGINEERING
THESIS SUPERVISOR

ACCEPTED BY .....
ORAL BUYUKOZTURK
CHAIRMAN, DEPARTMENTAL COMMITTEE ON GRADUATE STUDIES

BARKER

SOFTWARE DEVELOPMENT PROCESS: WEB-BASED PAVEMENT MANAGEMENT SYSTEM AS CASE STUDY

BY

WARIT DURONGDEJ

SUBMITTED TO THE DEPARTMENT OF CIVIL AND ENVIRONMENTAL ENGINEERING ON MAY 11, 2001
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ENGINEERING IN CIVIL AND ENVIRONMENTAL ENGINEERING

ABSTRACT

In the Information Technology Era today, software has been one of the most significant elements to help organizations achieve increased productivity and commercial success. For developers to create effective software, an appropriate development process must be applied.

Generally, the process of developing software can be considered as having six phases: requirements engineering, design, implementation, testing, maintenance, and project management. Over the past thirty years, different kinds of life cycle models have been developed by applying these phases to provide developers with the most appropriate procedures for projects of various types. In addition, a set of development fundamentals should be considered during the process to optimize time, effort and cost in developing each project.

This thesis studies the software development process and its effects on the development schedule of a Pavement Management and Inspection System (PMIS) project as a case study. From the case study, it can be concluded that choosing the appropriate life cycle model and applying the pertinent fundamentals, with the essential components of the development speed, can lead the project to be a success. Problems encountered during the development process are also valuable information to study as it may prevent them from occurring in the future.

THESIS SUPERVISOR: GEORGE KOCUR, PH.D
TITLE: SENIOR LECTURER, CIVIL AND ENVIRONMENTAL ENGINEERING

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor, Dr. George Kocur, for providing me the guidance and support to both my project and thesis. Without his advice and encouragement, this thesis could not have been accomplished.

I am grateful to Ron Santosousso, a senior engineer at the Arlington Department of Pubic Works (DPW). His patience and time spent in meetings, discussions and making recommendations to our team were very valuable to the success of the PMIS project.

Great thanks must go to my team members, William Cheung, Wesley Choi, and Anthony Yim. I will never forget the happiness and toughness that we have experienced together from this project and study at MIT during the past year.

Finally, I would like to express my cordial gratitude to my father, mother, and the rest of my family for their love and abundant support throughout my entire life. Very special thanks to P'Max and N'Fay, my lovely brother and sister, who have always given me strength and encouragement during my study at MIT.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The software development process provides a fundamental infrastructure for organizing and implementing software projects. This thesis will explore this development process and its effects on development schedules. A nine-month software project is described as a case study to capture knowledge about the development process based on actual experience.

1.1 Thesis Organization

This thesis consists of four chapters. These chapters are described as follows.

Chapter 1 provides an introduction to the thesis. It explains the contents of this thesis to give an overview to the reader of how this thesis is structured.

Next in Chapter 2, the software process and different life cycle models will be covered in order to help the reader understand the significance of each development phase to the software project. Also, at the end of the chapter, various software development fundamentals will be discussed to indicate how they contribute to a successful project.

Chapter 3 introduces the case study – the Pavement Management and Inspection System (PMIS), which is the focus of this thesis. The overview of this project is presented first, followed by the team organization and responsibilities. Then, the process of developing the PMIS system is investigated and the problems occurring during the process are discussed. Last, an overview and UML diagrams of PMIS are provided for a closer look at the system.

Finally, Chapter 4 discusses the essential components for a PMIS project, or any project, to be completed optimally and rapidly. Future enhancements for the PMIS project are also listed in this chapter as a set of suggestions to make the system more productive.

CHAPTER 2

SOFTWARE DEVELOPMENT PROCESS

This chapter explores the basic software development process and its effects on the productivity of the software project. To provide a good understanding of the software process, the term *software engineering* is introduced. Subsequently, the typical development process and life cycle models of the process are presented to describe phases that occur throughout the development procedure. Finally, development fundamentals, which should not be overlooked in the development process, are indicated as the factors to lead to an on time software project.

2.1 Definition of Software Engineering

The term *Software Engineering* was first reported in the first North Atlantic Treaty Organization (NATO) conference in 1968. In this conference, Professor Friedrich Bauer introduced software engineering as:

> *"The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines."*
>
> *(qtd. in Vliet 6)*

Since then, research on software engineering methodologies has been one of the major interests in computing science.

In the IEEE Standard Glossary Software Engineering Terminology, software engineering is defined as follows:

> *"The application of a systematic, disciplined, quantifiable approach to development, operation, and maintenance of software; that is, the application of engineering to software."*
>
> *(qtd. in Vliet 6)*

Despite the differences in software engineering definition, the basic objective is essentially the same: to develop the methods and procedures for large-scale software with high productivity at low cost, controllable quality and measurable development schedules.

2.2 Typical Software Development Process

When constructing software, the problem to be solved is first analyzed, and then requirements are described in a very precise way. Subsequently, a design is prepared based on these requirements. Finally, the construction process, which is the actual programming, is started. The phases of a common software development process can be depicted in Figure 2-1.

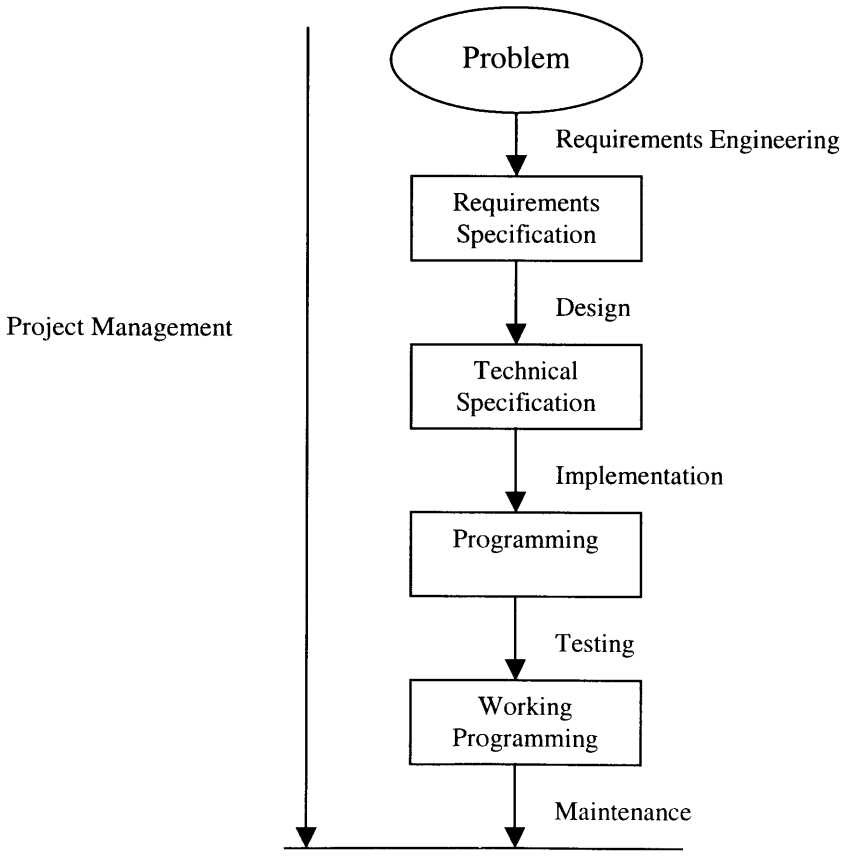


Figure 2-1: A simple view of software development process (Vliet, 2000)

The process model in Figure 2-1 is rather simple and has been depicted sequentially. For a given software project, these activities are not necessarily separated as strictly as indicated here. They may and usually will overlap. For instance, it is possible to start the implementation of one part of the system while some other parts have not been fully designed yet. (Vliet, 2000) Below, a short explanation of each of the basic components is given.

2.2.1 Requirements Engineering Phase

The goal of requirements engineering is to get a detailed description of the problem to be solved. The document in which the result of this activity is recorded is called the *requirements specification*, which is what this phase is trying to achieve. A feasibility study may also be performed here for assessing whether there is a solution to the problem. Note that care must be taken during this phrase in order to obtain a final system that will meet customer expectations. (Vliet, 2000) This implies that there must be an effective communication system and collaboration between various groups of people such as the customer, prospective users, and designers.

2.2.2 Design Phase

The purpose of the design is to create the solution to a problem specified in the requirements specification. The structure of the whole system is developed to solve the problem for the customer. It is imperative to understand that the design process is to focus on *how* to satisfy the customer needs as opposed to the requirements engineering phase, which concentrates on *what* is needed for the system. The *technical specification* is the output of the design phrase and serves as a starting point for the next step, which is the implementation phrase. (Vliet, 2000)

2.2.3 Implementation Phase

This phrase is also known as the programming phrase. The conversion of the specification into the *executable program* is what the implementation phase seeks to

achieve. (Vliet, 2000) Note that the main goal of a developer here is to produce a well-documented, reliable, easy to read, flexible and correct program in preference to a very efficient program that is full of tricks. Furthermore, in order to understand the design and produce final executable codes, a developer must remain in close contact with the designer.

2.2.4 Testing Phase

Testing's objective is to uncover requirement, design and coding errors in the program and to ensure that defined input will produce actual results that agree with required results. Despite it being the fourth phase, testing should actually be carried out at all stages. The earlier that errors are detected, the lower the cost and effort spent to correct them. Testers will then prepare the final *test report* and *error report* at the end of the testing phrase to make sure that the software works properly as planned. (Pressman, 2000)

2.2.5 Maintenance Phase

This is an important phrase as, after the final system is delivered to a customer, there may still be some undetected errors and/or the software may need to be improved. The maintenance phrase basically focuses on changes in the software. This phrase reapplies all the steps discussed above (i.e. from Section 2.2.1 to 2.2.4) for existing software instead of creating new software. Therefore, this phrase can be seen as what is needed to keep the system working. (Vliet, 2000)

2.2.6 Project Management Phrase

This activity will span all phases in order to ensure that the final product is delivered on time and within the budget. The project management team is responsible for planning, team organization, quality issues, and cost and schedule estimation. (Vliet, 2000)

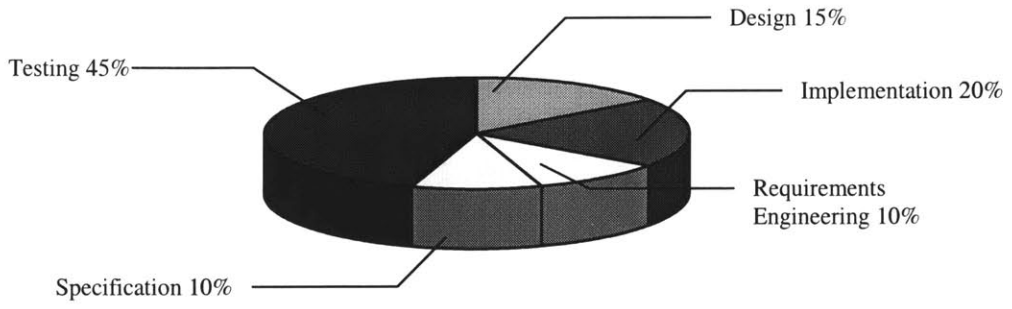


Figure 2-2: Relative effort for each phrase of software development (Vliet, 2000)

Figure 2-2 above demonstrates the relative effort spent on the various activities up to the delivery of the system. From this data, a clear trend emerges which is called 40-20-40 rule: only 20% of the effort put in is spent on actually implementation (programming) the system, while the preceding phrases (requirements engineering and design) and the testing phrase each consume about 40% of the total effort. However, this is not the only rule to be considered; the effort of each phrase also depends on the properties of the software to be developed, particularly the size of the project. These reasons also imply that other combinations (such as 60-15-25), although less implemented, can be used. Despite this, the 40-20-40 rule is widely used in most projects (Vliet, 2000).

2.3 Software Life Cycle Models

The Life Cycle Model is a prescriptive model that specifies what activities should happen in each phase and establishes the order in which a project is performed. The model also sets up the criteria that will be used to determine whether to proceed from one activity to another. The nature of the project and application, the methods and tools to be used, and the controls and deliverables that are required are critical factors that will determine an appropriate model to be used. The discussion below describes three process models and outlines the corresponding advantages and drawbacks of those models.

2.3.1 The Waterfall Model

The waterfall model is essentially a slight variation of the model in Figure 2-1. It is one of the oldest widely used models for various software projects nowadays and sometimes is referred to as the *linear sequential model* or *classic life cycle*. The waterfall model was first described in 1970 by Dr. Winston Royce. It suggests a systematic, sequential approach to software development that begins at the system level and progresses through requirement analysis, design, implementation, testing and maintenance. Figure 2-3 illustrates components of the waterfall model.

It is important to note that the output obtained at the end of each phase will be compared with the projections. The *Verification and Validation* process is to perform this comparison. Verification is used to check if the system meets project requirements and therefore strives to assess the correctness of the transition to the next phase, while validation checks if the system meets the customer's requirements. (Vliet, 2000)

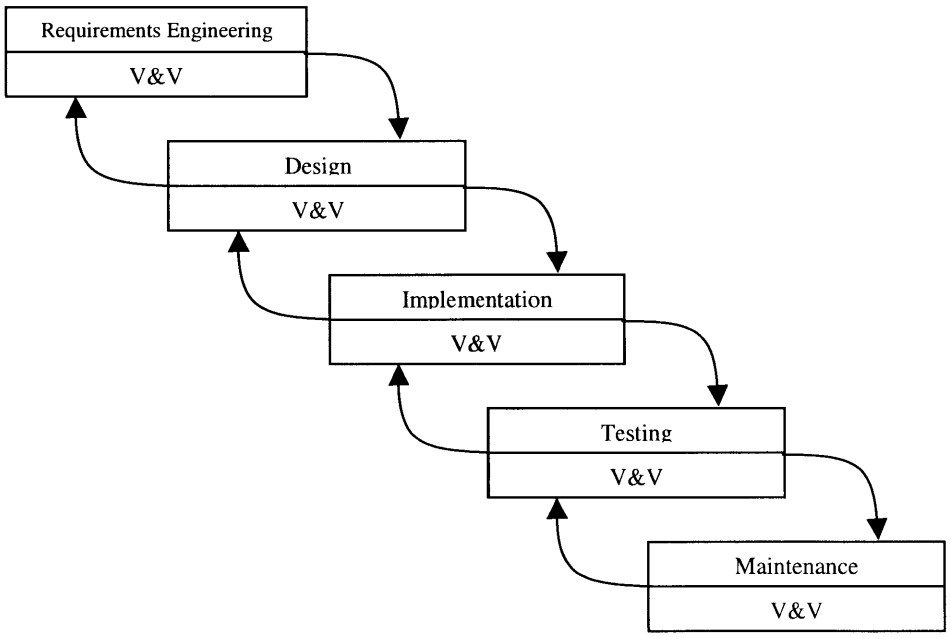


Figure 2-3: The waterfall model (Vliet, 2000)

Note: V&V stands for Verification and Validation

There are several advantages of applying this model to a software project. First, the linear ordering of activities as well as the verification and validation process described above help to detect errors at the early and low-cost stage of the project. Also, it renders the development process more structured and manageable as the progress can be traced more easily and accurately to discover possible delays. (Pressman, 2000) It has been suggested that the waterfall model works particularly well if there is a technically weak staff or an inexperienced staff simply because the model provides the project with a structure that assists to minimize wasted effort (McConnell, 1996).

However, the waterfall model has its limitations. First of all, the difficulty of specifying all the requirements by the customers at the beginning stage is one of the most commonly cited problems. The model requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects. In addition, the linear ordering of the model requires that each stage has to be completed before the next phrase can begin. This order may not be the case in reality as actual projects rarely follow the sequential flow that the model proposes (Pressman, 2000). Moreover, the tangible results in the form of software may not be available to a customer until the end of a project life span and therefore, a major blunder, if undetected until the working program is reviewed, can be disastrous. Last, this model takes a long time to deliver the first release, which may not be commercially acceptable or, even if it is, the business requirements will often have changed during the development period, so that the system no longer meets the business needs.

2.3.2 The Prototyping Model

As stated in the previous section, it is difficult, if not impossible, for a customer to specify all of the software requirements at the beginning of the project, and this makes the waterfall model difficult to implement in practice. In such cases, the development of one or more prototypes may overcome this difficulty.

Prototyping is illustrated in Figure 2-4. It consists of many different phrases and is divided into two major stages; the prototyping stage and the actual production stage. The fact that all processes are developed and progress much more quickly with lower costs in the prototyping stage is what makes this stage different from the latter. (Vliet, 2000)

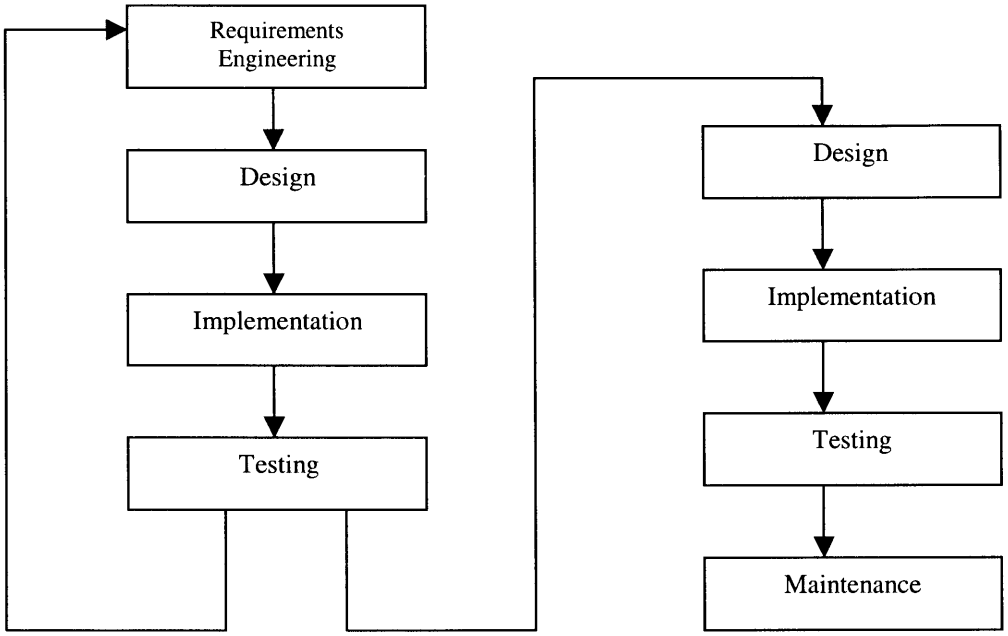


Figure 2-4: The prototyping model (Vliet, 2000)

In the first stage, the process begins with getting the requirements from the customer. A quick design is the next step, which is then followed by the implementation and testing phrases. Once these are completed, a prototype is created for the customer to view and comment on.

It can be seen from Figure 2-4 that at the end of the first stage, if the prototype has not satisfied the customer, it is then discarded. The process does not continue to the production phrase; in contrast, it goes back to the starting point – requirements engineering. This is known as *throwaway prototyping*. However, if customer satisfaction is reached after viewing the first version of the prototype, the raw requirements is formulated based on its functionality. Then, the production phrase begins and creates the next version of the prototype. After several iterations are carried out and the customer is

fully satisfied with the software, the last developed prototype version is the final system and this is known as *evolutionary prototyping*. Generally speaking, *throwaway prototyping* occurs much more often than the latter. (Vliet, 2000)

Ideally, prototyping is a mechanism for the requirements engineering phase. This is because after the prototype has been created, it will be evaluated by the customer and used to refine the requirements for the actual software to be developed.

There are three main advantages of implementing this model in the software project. Most of all, the end result is a leaner system whose functionality matches the real user requirements. This is because only the key requirements have been specified at the beginning, as there is no need for the customer to express all requirements up front. In addition, if the customer has the influence to modify the design and features, the system features will better reflect the requirements and the system will be easier to use. (Pressman, 2000)

Since the working system can be tried out continuously, it helps developers to detect some of the problems that could arise in the early stage. This is another advantage of utilizing the model as it prevents the waste of effort that would otherwise be needed to redo part of the work. (Pressman, 2000)

Last, given the fact that the steps involved in this model will be iterated, it will often make the quality of the final system higher and hence easier to maintain.

Despite these advantages, prototyping can also be problematic for the following reasons. When the customer sees what appears to be a working version of the software, unaware that in order to get it working quickly, no one has considered overall software quality or long-term maintainability, there is a perception problem. Even when they are informed that the product must be rebuilt in order to achieve and maintain high levels of quality, they often demand that *a few fixes* be applied to make the prototype a working product and software development management often relents. (Pressman, 2000)

Another reason relates to the *less than ideal* choices made by the developers to prototype the overall system. The developer often makes implementation compromises in order to get a prototype working quickly. An inappropriate operating system or programming language may be used simply because it is available and known to them. After a while, the developer may become familiar with these choices and forget reasons as why they were inappropriate. (Pressman, 2000)

It is important to note that even though problems can arise, prototyping can be an effective model for software engineering. The key to correct these mistakes is to ensure that the customer and developer agree that the prototype is built to serve as a device for defining requirements. Overall, prototyping works well with an experienced team as it involves making the far-reaching designing decisions, especially during the iteration process in which user requests must be weighed against the ease and cost of realization.

2.3.3 The Spiral Model

From the prototyping model, it is helpful to consider software development as an iterative cycle in that each cycle includes the same phases; requirements engineering, design, implement and testing. The spiral model, proposed by Boehm, is a popular evolutionary software lifecycle model that applies both the iterative and systematic nature of other models with processes moving around a spiral as shown in Figure 2-5.

The spiral model can be seen as a risk-oriented model because major risks will be minimized in each cycle of the model by the risk analysis process. Once these risks have been addressed, the spiral model ends in the same way as a waterfall model does. Risk in this case is referred to a poor understanding of the requirements, design or problem in the underlying technology. (McConnell, 1996)

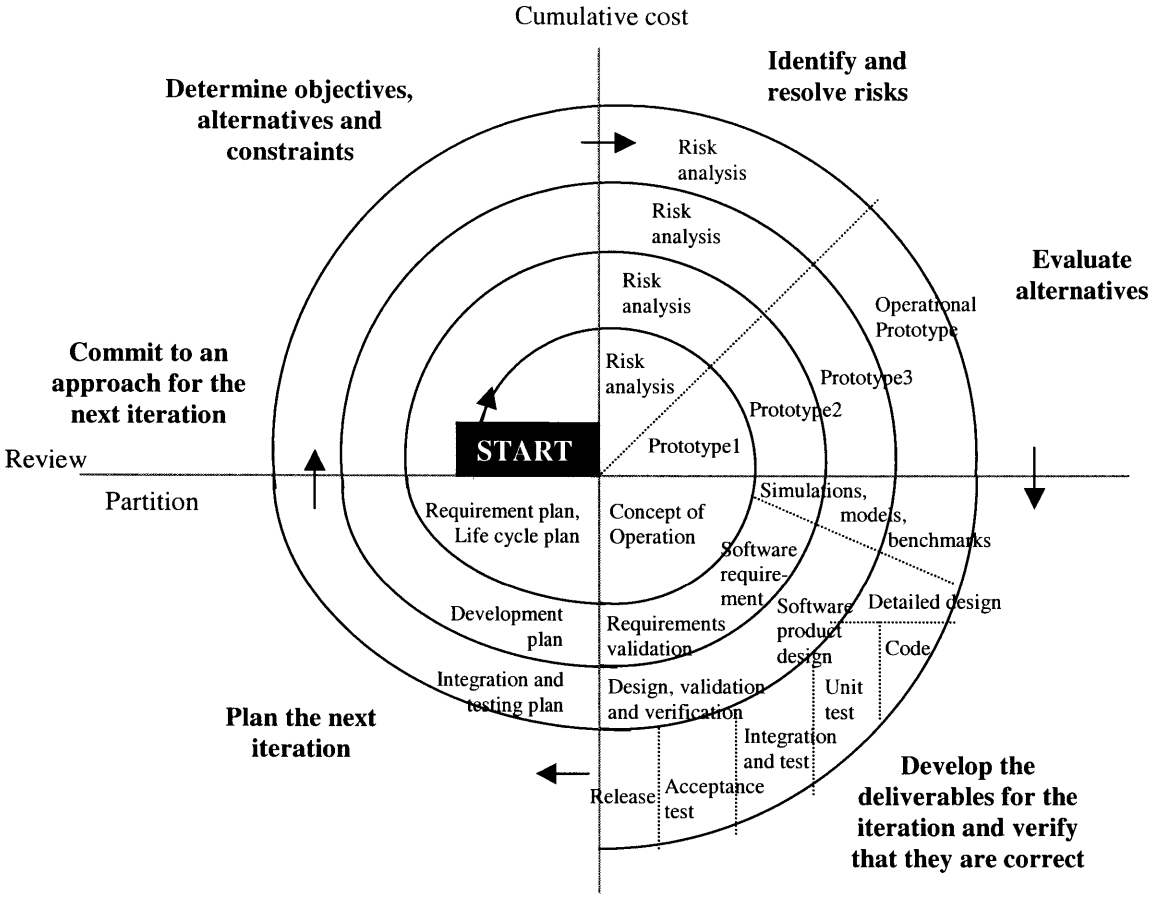


Figure 2-5: The spiral model (McConnell, 1996)

As Figure 2-5 illustrates, software development starts at the core of the spiral with a small-scale project. It first examines the risk, and then develops a plan to overcome those risks and commit to an approach to the next iteration. Six steps of the iterative process are listed below.

1. Determine objectives, alternatives, and constraints
2. Identify and resolve risks
3. Evaluate alternatives
4. Develop the deliverables for that iteration and verify that they are correct
5. Plan the next iteration
6. Commit to an approach for the next iteration

Some view the spiral model as a progression of the first two models discussed earlier. As can be seen from Figure 2-5 that the model has both the waterfall model and prototyping properties in itself. If getting the accurate requirements is considered to be the highest risk, following the spiral model several times can fix this problem (using the prototyping iterations). Alternatively, if the detailed requirements are known at the starting point, and the aim is to obtain an effective and well-documented system, following the spiral model and finishing with the waterfall model can help reaching this goal. (McConnell, 1996)

The detection of risks at early stages of software development is the most important benefit of the spiral model. As the process progresses, both developers and customers have a better understanding of the system and know how to react to the risks at different cycles. In addition, the more time and effort spent on the project, the less risk is taken and that will bring the project to the rapid development speed. (McConnell, 1996)

However, the complexity of the spiral model is the major pitfall. The model requires conscientious and advanced management skills as occasionally it is difficult to define the objective and provable milestones indicated whether it is ready to move forward to the next cycle. Besides, in projects where the development processes are straightforward and the risks are modest enough, the risk analysis activity may be redundant and need not be performed in each cycle. (McConnell, 1996)

2.4 Software Development Fundamentals

Even when the software life cycle model has been chosen appropriately for a project, if the basic software development fundamentals are ignored and/or not properly applied throughout the project, the result is likely to be a delay in the software delivery and a more costly project as there will be more time and effort spent in complete the project. Therefore, basic fundamentals must be considered during the development process in order to deliver well-designed software on time.

The fundamentals of software development are generally divided into three parts: technical, management, and quality assurance. This section discusses each of these fundamentals and explains the effect of software development fundamentals on development schedules. (McConnell, 1996)

2.4.1 Technical Fundamentals

A study of *Modern Programming Practices* in 1984 found that developers could not achieve high productivity without using them. However, using a high-level programming language or advanced technology does not necessarily result in high productivity. (McConnell, 1996) As indicated in Figure 5-6, even though the use of advanced programming practices is high, low productivity could still occur.

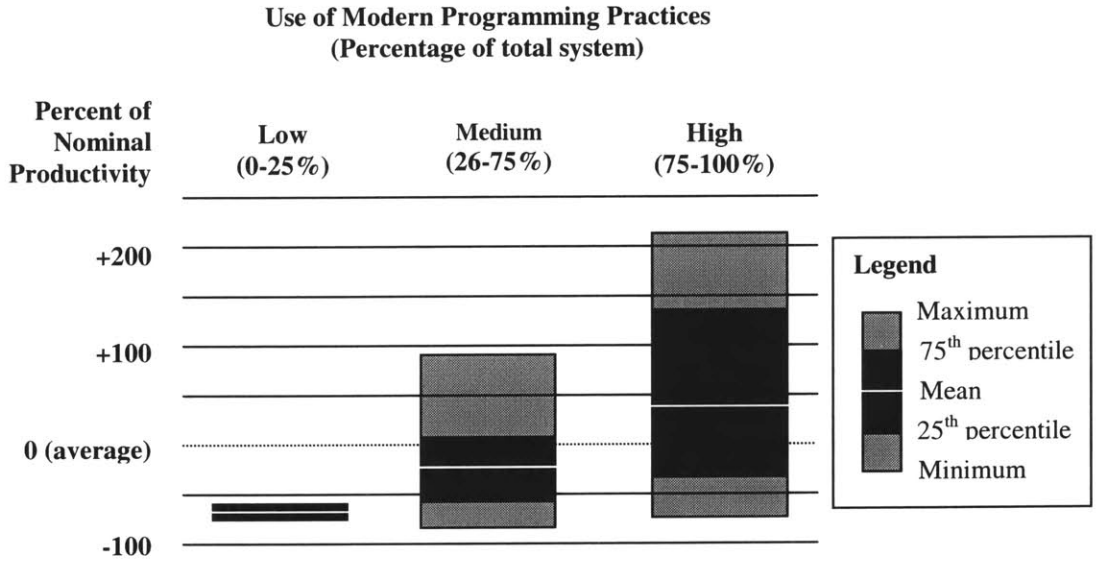


Figure 5-6: Use of Modern Programming Practice (McConnell, 1996)

The following section discusses various technical processes so that rapid software development can be achieved.

Requirements Management

Requirements management is a process of acquiring the requirements, tracking the design and code, and continuously improving them until well-designed software is launched. A

lack of user input, or inadequate/changing requirements are examples of incompetent requirements management practice that, if they arise, will often result in delay in software delivery, and project costs exceeding the budget.

The success of requirements management depends on the ability to choose an appropriate fundamental approach for a particular project. These requirements management fundamentals include requirements engineering methodologies, system-modeling practices, and the relationship between requirements management and lifecycle models. Requirements management can greatly facilitate software development speed when gathering requirements is done quickly without compromising the quality. (McConnell, 1996)

Design

It is possible to develop software without designing it first. However, design serves as a foundation of programming, project scheduling, project monitoring and project control, and effectual design is important in assisting the speed of development. The fundamental topics in design include design styles, foundation of design concepts, design approaches for the challenging areas, and use of design tools. (McConnell, 1996)

Implementation

Although implementation practices do not offer as great leverage in the development schedule as requirements management and design, they still deserve some attention. Ignoring implementation fundamentals could result in insufficient code quality, and it generally takes more time and effort to redo coding, resulting in a delay in software delivery. Furthermore, bad implementation practices can create subtle errors, and again consume a large amount of time to uncover and rectify them. Examples of implementation fundamentals include coding practices, data-related concepts, control-related concepts, unit testing and debugging practices. (McConnell, 1996)

Software Configuration Management

Software configuration management is a practice of controlling the project artifacts so that the project remains in a stable state over time. It includes practices such as assessing the proposed changes, tracking changes, handling multiple versions and keeping copies of the software artifacts existing at different times. Lacking of configuration management when there is an amendment to the design could lead to a lengthy recoding process. For example, without configuration management, a software developer may not know about design changes and may implement coding that may not be compatible with the new design making rework necessary. (McConnell, 1996)

2.4.2 Management Fundamentals

One may see technical fundamentals in the previous section as the most important influence on the development schedule. Management fundamentals, however, also play an important role and quite often, have at least the same influence as technical fundamentals, if not more. (McConnell, 1996) Typically, there are three aspects of the development, which are affected by management fundamentals. These are schedule, cost and product. Components of management fundamentals are estimating the size and scheduling the time in the project, allocating the resources appropriate for a product size, creating a plan for the project and, monitoring and directing the project to ensure that it is on the right track.

Estimation and Scheduling

To create a software schedule, three basic steps must be performed. First, the size of the project is estimated, then the effort needed for the project is approximated based on its size and finally, the estimation of schedule is performed based on its effort. The estimation and scheduling is vital to an efficient software development simply because accurate estimation leads to effective planning. (McConnell, 1996)

Planning

After the estimation is completed, planning should take place mainly to select project members and determine the appropriate skills required and the organization of the team. Depending on the nature of the project, a lifecycle model will be chosen accordingly for software development. (McConnell, 1996)

Monitoring

Without monitoring, it is difficult to recognize whether the project is on the right track. Monitoring seeks to ensure that the project meets the scheduling, costing and quality plan. It essentially consists of two parts. One relates to the management-level monitoring controls and the other relates to the technical-level monitoring controls. The former can be tracked by using the task lists, status meetings, milestone reviews and report, while the latter can be tracked by doing technical. Effective monitoring enables project schedule problems to be detected and fixed in early stages. (McConnell, 1996)

Measurement

Measurement is the activity of collecting data for the purpose of analyzing software quality and productivity. It serves as an important key to the long-term progress for the software organization, as it provides rough guidance for estimating and scheduling of future projects. Quite often the collected data relates to project costs, schedules, and size in terms of the lines of code. (McConnell, 1996)

2.4.3 Quality -Assurance Fundamentals

It is pertinent to note that software developers should not compromise software quality by ignoring quality-assurance fundamentals when there is a tight deadline. Quality-assurance fundamentals will in fact assist software development speed as they help to detect defects and remove them in the early stages of the project and hence, increase the speed of software development. These fundamental practices can be performed either by testing or technical reviews.

Testing

Testing is a common practice to detect errors by executing the coding. There are two kinds of testing, unit tests and system tests. Unit tests are performed by the developers who verify that the coding works correctly, while system tests are performed by testers who ensure that the entire system operates as desired. (McConnell, 1996)

Technical Reviews

Technical reviews aim to detect errors in the requirements, design, implementation, or other project artifacts. The most common types of technical reviews are walkthroughs, code reading and inspections.

Walkthroughs take place when the developers review their technical work and make any necessary adjustments to improve software quality. This process will effectively maximize the speed of development because it can detect errors in the requirements before the design or any coding is performed. This is more appealing than testing as testing only detects errors once the implementation (i.e. coding) is completed. (McConnell, 1996)

Code reading is similar to walkthroughs but only applies to the code. The code will first be read and any detected errors are then reported to the developers so that they can correct them. It is suggested that code reading should be performed together with testing in order to achieve more effective software and increase the speed of software development (McConnell, 1996).

Inspections are formal technical reviews and are very effective practices in detecting any errors throughout the project. In this process, the developers will form a team and each of them will be responsible for either one of the following roles – moderators, reviewers, and scribes. The *Moderators* distribute the work that needs to be inspected to the *Reviewers* who then examine the work before the meeting. During the meeting, the reviewers report detected errors while *Scribes* record those errors. The moderator then produces the inspection report that explains each error and identifies the solution.

Inspections, like walkthroughs, can detect errors much earlier than testing can. (McConnell, 1996)

It can be seen from this chapter that the software development process should be considered and clearly planned before initiating the software project. Both life cycle models and software development fundamentals must be chosen and applied appropriately throughout the project in order to achieve success in developing the software project.

CHAPTER 3

CASE STUDY:

PAVEMENT MANAGEMENT AND INSPECTION SYSTEM

To study the software development process in more detail, a Pavement Management and Inspection System (PMIS) has been applied as a case study. PMIS was created by a group of Master of Engineering students in Information Technology to obtain a practical knowledge of developing a software system. The basic software development processes and techniques were used throughout this project.

In this chapter, the overview of the PMIS project is presented first and is followed by discussions of the project development team and its development process, as well as the problems encountered during the project. Last, the PMIS system and the unified modeling language (UML), including use case diagrams and sequence diagrams, used to develop the system are described.

3.1 PMIS Overview

3.1.1 Problem Statement

The town of Arlington has an area of approximately five square miles and has about 100 miles of roads and streets. In order to update the Arlington street database and estimate street conditions, the streets must be inspected and recorded on a regular basis. The decisions on maintenance and repaving plans are made based on the street conditions that are currently represented by a *PSI* value (*Pavement Serviceability Index*), which takes a value of between zero (poor condition) and five (good condition). The annual budget is based on an analysis of which streets have the largest maintenance needs. The town now

has only a ten-year-old DOS based-system to implement the analysis. This system is called the *Integrated Management System, IMS*, and is a pavement management system. The inability to fully meet the current needs, and the difficulty of using IMS due to its complicated user interfaces, are the main pitfalls of this system. Furthermore, it requires an extensive amount of inspection data to evaluate the current street condition, which usually results in unnecessary time and effort spent on information gathering during the inspection process.

The *Pavement Management and Inspection System, PMIS*, is a Web-based system that has been developed in order to remedy the defects currently existing in IMS. PMIS provides the same general functions as IMS, but is designed to be more convenient to use and provide better decision support tools. In addition, in order to help the inspector reduce the time spent during the inspection process, the method for calculating the street conditions was revised to require less data than the existing system. A Personal Digital Assistant (PDA) such as a Palm®, and a Global Positioning System (GPS) unit are used to record inspection data and indicate street position, which are transferred to the PMIS system.

3.1.2 Background

PMIS is a nine-month development project that began in September 2000 and is due to be completed in May 2001. The development team consists of six students, an advisor, and one senior engineer from Arlington.

The advisor and students typically meet once a week to discuss the project, as well as to review problems that might have occurred during the process. The senior engineer and students, on the other hand, meet approximately twice a month to gather the necessary information for designing and developing PMIS, including discussing system development progress. In this project, many development practices and tools, such as software lifecycle models, CVS and other tools in the Linux operating system, the Java programming language environment (including Java Server Pages or JSP, Java Beans,

JDBC database connectivity) and the MySQL database management system, are utilized extensively.

3.1.3 Objectives

Throughout the development of PMIS, students are expected to apply software development knowledge to the project and to create a real working system. They learn about the software development processes from the beginning (i.e., collating information and converting it to fit the system requirements) to the end (i.e., delivering the final product to the client). Moreover, students are able to improve their problem solving skills due to the time constraints, problems and mistakes that can arise during the development process. Finally, as the project is very team oriented, students learn how to collaborate effectively among themselves in order to achieve the optimum result.

3.2 PMIS Organization and Responsibilities

PMIS is organized to enhance team performance and achieve the project's objectives. Each team has its own goals and task, and consists of a different number of students, who are given the opportunity to choose their roles based on their knowledge and interests at the beginning of the project.

Six members are grouped into three separate teams. The professor who acts as a project advisor provides guidance and assistance to the team members. The PMIS organization includes a senior engineer from the Arlington Department of Public Works (DPW), whose role includes giving suggestions about how to improve the system and which features should be included in the PMIS. However, after the project had commenced, there was a demand to reorganize the teams. This change was to ensure that each member in the teams had the interest in their responsibilities. The reason for requesting the team reorganization will be described later. The final team organization and team members of PMIS are depicted in Figure 3-1.

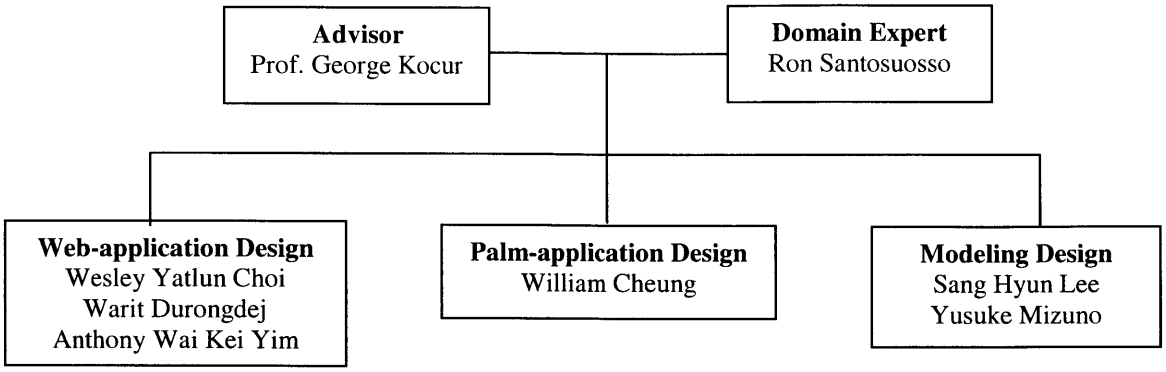


Figure 3-1: PMIS Team Organization

As shown in Figure 3-1, the PMIS project is comprised of three teams: *Web-application design team, Palm-application design team*, and *Modeling design team*, which all have different responsibilities. The following discussion briefly explains the tasks of each team.

3.2.1 Web-application Design Team

Web-application designers are mainly responsible for creating a new Web-based system to replace the old IMS. Since the purpose of the new system is to increase the ease of use of the system and the customer's needs, user interfaces need to be improved, new features need to be added, and the system's database needs to be revised, so as to enhance the system efficiency. In order to fulfill this goal, the team must collect all crucial information from the customer and convert it to the system requirements. The meetings with the customer and the senior engineer are held at the DPW in Arlington usually to gather all the information important to creating the new system. After finishing the information gathering, the team must design the features and define the main functions of PMIS from the requirements. The requirements and design specification are then used to provide a reference for implementation (programming) and testing. The PMIS executable code is then generated to make the system work. The team must create concise, high quality and documented code using the Java environment, in order to help the testing procedure at a later stage.

3.2.2 Palm-application Design Team

This team is responsible for creating the application to be executed on the Palm PDA that can communicate with both the GPS and the PMIS system. This application will be used in the inspection process to collect defect data of streets. The Palm will be used to record those inspection data instead of writing them on a sheet of paper, whereas the GPS will be used as a tool to determine the position of streets. In the same way as the Web-application design team, the member of this team has to create the requirements and specification for the Palm application by collecting the relevant information from the customer. The Palm-application designer must then design and generate the code for the Palm using the C++ programming language.

3.2.3 Modeling Design Team

The modeling designers are responsible for revising the method of determining the street conditions for the PMIS. The team must apply transportation engineering technical knowledge to simplify an existing approach so as to reduce inspection data involved in both calculating the present street conditions and predicting them in the future. A method for estimating the cost-benefit ratio of each maintenance action is also generated in order to provide the customer with the most useful and effective information. The team members then develop a method to calculate PSI, costs and benefits. Likewise, the requirements and specifications for the new methods of calculation need to be developed and the executable code needs to be written using the Java programming language.

Once each team has completed its tasks, system testing will begin, which usually requires the cooperation of each team. The objective here is to ensure the integrity of the whole system by finding any possible errors in the programs. The procedures that are used to test the system are code reading and walkthroughs, which are mentioned in Section 2.4.3. The iteration of the implementation and testing phase may be performed again to correct any mistakes and ensure that the programs run correctly. Finally, a PMIS manual or a user guide document is created for the customer to effectively utilize the system.

3.3 PMIS Process

3.3.1 PMIS Development Life Cycle

Based on the discussion of Team Organization and Responsibility above, the PMIS development life cycle is depicted in Figure 3-2. It can be said that this development process contains characteristics of both the *Waterfall Model* and the *Spiral Model* as discussed below.

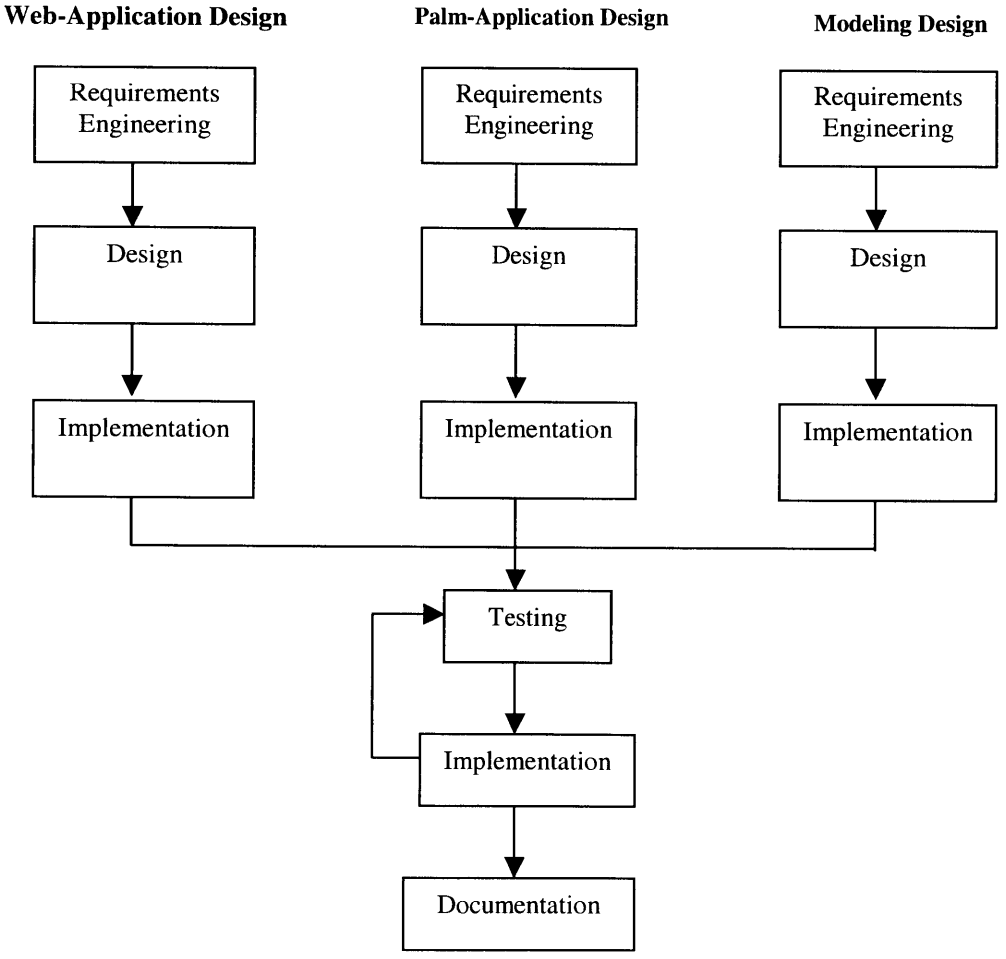


Figure 3-2: PMIS Development Life Cycle Diagram

The waterfall model was applied to the project during its early stages, which were the requirements engineering phase and the designing phase. The reason that this model was used is that the customer could state most of the requirements up front, since he had a deep understanding of what was needed to improve and include in the new system. In addition, as stated earlier, changing from the old IMS DOS-based application to the new PMIS Web-based application required the redesign of the user interfaces, the system's features and database, therefore applying this model to these phases made the project go quickly, so that the customer can see the output of each stage and could check and comment on the system's requirements and specification.

The spiral model was, on the other hand, applied to the later stages of the project (i.e., the implementation phase and the testing phase). This model was chosen because each member of the PMIS teams had limited experience in both the computer programming language and software development methods used (which was considered to be a crucial factor in the project). Applying this model to the later stages of the PMIS process helped to detect mistakes/errors and reduce risk, which could have arisen from the lack of understanding of both the technical and PMIS system issues.

Note that there might still be some mistakes and/or errors in the requirements, specification, and/or executable code after the project ended. However, performing the iteration over and over again could eliminate this. The ideal situation would be to apply the spiral model to the whole process, i.e., from the requirements engineering to the testing stages, but due to the time constraints (with the project being nine months long), the team members had to complete the project once the second iteration of the testing and implementation phases were performed.

3.3.2 PMIS schedule

After the responsibilities were assigned to each of the three–teams' members, the project schedule was planned as described below.

September 2000 - December 2000

Throughout the fall term, all PMIS teams concentrated on creating the requirements and the specifications of the system. Furthermore, the hardware used as workstations for the project and a server for running the system were also set up during this period. The completion of these activities was in mid December 2000.

January 2001 – March 2001

Each PMIS team started the programming stage in mid January 2001. There was a pause of the developing process after finishing the requirements and specifications and before beginning programming. This time was spent studying the computer languages and technology that were used for programming. The first programming period finished at the end of March 2001.

April 2001

In this month, the focus was on testing the executable code that had been produced, in order to detect problems or mistakes in the system. This required each team to do a second programming phase to rectify the code and make the system work correctly.

May 2001

Documentation is also important for the end user as it will give guidance and/or explain to the customers how to utilize the system. Therefore, the process terminated with the PMIS teams creating a user manual, which describes the characteristics of the PMIS system.

3.4 Problems to the PMIS Development Process

Throughout the development process of the PMIS system, there were various kinds of problems that arose and resulted in a decrease in the development speed. These problems should be carefully studied to prevent similar mistakes from happening in the future. They can be classified into two categories: individual and team problems.

3.4.1. Individual Problems

An individual member is important to a project, as the individual is the essential force that makes the project proceed. A project that has fewer but more proficient members can be developed at a faster pace than one with incompetent, though more numerous, members. The major individual problems to be taken into account in the PMIS project are motivation and ability.

3.4.1.1 Individual Motivation

The motivation of members in a team is a significant factor in the speed of the project, since having high motivation to perform their work can make the project progress rapidly. Lack of motivation can result in delay of the development process and it was the very first problem that was encountered in the PMIS project.

At the beginning of the project, all the team members had little software development knowledge, as most of them had no prior experience in the software development industry. When they had to choose the roles that they were responsible for in the project, some of them did not fully understand what responsibilities of each role were. Such a situation can undermine the members' motivation to fulfill their roles and that happened in this project.

More confusion and dissatisfaction came up when the members began their work, which resulted in requests for a change in team organization. Once the change had been made, some members left the team and other members joined the team. The new members needed to catch up with the work that had previously been done by the members who left, or in the worst case, they had to redo it. Either way, it delayed the project even further. However, this situation primarily occurred in September and October, which was the beginning phase in the development process. After things had settled down, each member knew well of his responsibility in the project and had the motivation to work on what he was interested in and had chosen to do.

3.4.1.2 Individual Ability

Individual ability plays an important role during the development process of a project. Without adequate technical skills, a member cannot successfully implement the project that had been specified and planned for the customer.

The problem in this project was that all team members had their primary backgrounds in Civil Engineering with but little background in Information Technology. Despite the fact that some members had knowledge of several computer languages, most had no experience or familiarity with the technologies that were used to develop this system (i.e., Java Server Pages, MySQL, Palm and Linux OS).

However, these individual weaknesses were corrected by letting each member spend time (a month after the fall semester as indicated in the PMIS schedule) to study and practice these technologies. At the beginning of the implementation phase, the progress of the project might have been slow because these technical skills had to be improved from experience, but it increased in speed when all members started to get more acquainted with the technologies.

3.4.2 Team Problems

The number of members and teams in a software project increases when the project scope (as defined by requirements and design) grows bigger. This makes the project much more difficult to manage and can lead to problems of collaboration (between team members) and coordination (between teams).

3.4.2.1 Team Collaboration

The collaboration of each member in a team is a vital issue to the pace of a project. If each member can get along and work very well with the others in the same team, even a complex project can be developed efficiently. Lack of smooth collaboration can result in delays from each team, and this will affect the speed of the entire project.

For the PMIS project, even though there were no personal problems between team members, their collaboration was not at an adequate level as can be illustrated by the following example. Throughout the development process, there were only a few formal meetings set up for each team to discuss its work progress or the problems of members in the project. Therefore, team members sometimes did not understand the others' ideas and suggestions, or know what the others' assigned tasks were. As a result, there was some uncompleted work left and some redundant work done. Both of these circumstances required members to devote extra effort that should have been unnecessary.

However, as stated in Section 3.1.2, meetings were held generally once a week with the advisor to discuss the project's improvement, and these meetings reduced confusions of some of the team members. In addition, since the friendship between the team members gradually grew after they had been grouped together, casual communication among them helped them convey their ideas and/or suggestions to others, even though formal meetings of team members were not established.

3.4.2.2 Team Coordination

Besides the collaboration of the team members, coordination between teams is also another key to the development speed of the project. Without coordination between each team, the project cannot be carried out smoothly.

The following example demonstrates a situation in which lack of coordination occurred in the PMIS project delayed the development speed. When the Web-Application design team and the Modeling design team were designing their parts of the PMIS database, they had not consulted each other before creating entities and tables that they both needed to use throughout the project. Accordingly, there were some conflicts between the designed tables when they combined them. This led both teams to fix their designs in order to make the database correct, consistent, and easy to use. This unnecessary work, even though it was not large in amount, caused the project to slow down for a while.

Both kinds of team problems (collaboration and coordination) that came up during the PMIS development process could be addressed by assigning some individuals to be team leaders and a project leader. These leaders would be responsible for managing members and tasks in each team (for team leaders) and between teams (for the project leader) to ensure that there was no redundant work and that the work that needed to be done was done and in the correct way. However, in the PMIS project, there were no volunteers to play these roles.

3.5 Summary

At the end of the nine months, with the knowledge and efforts that each PMIS member that had contributed to this project, the PMIS system with its performance and efficiency that satisfies both the team members and the customer was delivered. It provided most of the functions that had been specified in the system's requirements. The final system did improve its user interfaces, along with added new functions for conveniences of the customer according to the purposes. The Palm PDA and GPS were fully designed and developed to serve as equipment for the inspection process. By connecting them together, a user can record the inspection data by simply clicking on the Palm's screen, which helps to decrease time spent during this process.

Even though various problems occurred during the development process of the PMIS project, these problems were considered to be very typical for all software projects. However, if they could be avoided, a project would be more effectively accomplished. For non-professional developers, such as students, the amount of mistakes and errors were at an acceptable level. The experience and problems that every member of the teams obtained from the PMIS project will be very useful for developing future projects.

3.6 PMIS System Introduction

The new PMIS system for pavement management emphasizes ease of use for a user. The system has been improved from a DOS-based to a Web-based system to eliminate complications in using the system. In addition, as being the Web-based system, multiple users are allowed to work on the system from different locations. This section introduces PMIS system including screenshots and descriptions to help the user gain a better understanding of using it. This PMIS introduction is described in order of processes and the functions provided in the system.

3.6.1 PMIS Home Page

Once a user goes to the main page of PMIS, the system provides the function to check the status of the user to determine whether he has the permission to access the system or not. For an existing user, he needs to fill in his username and password. Figure 3-3 depicts the system's home page and the login function.

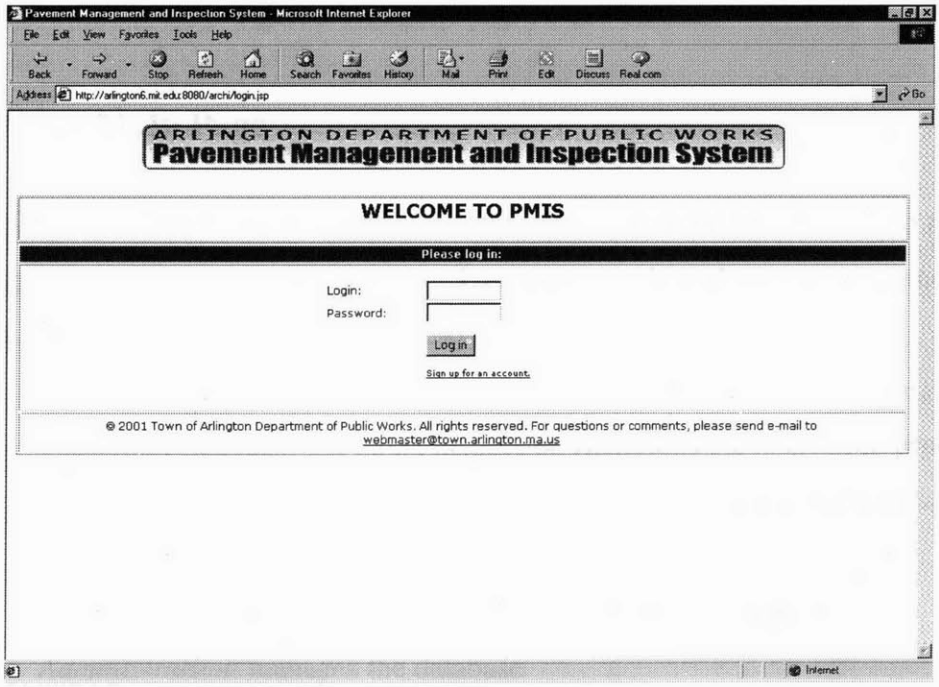


Figure 3-3: PMIS home page

For a new user, he can register for a new account by giving information including his username, password, first name, last name, department, position and e-mail address, as in Figure 3-4 below. Then, the system creates a new individual profile stored in the database for the user.

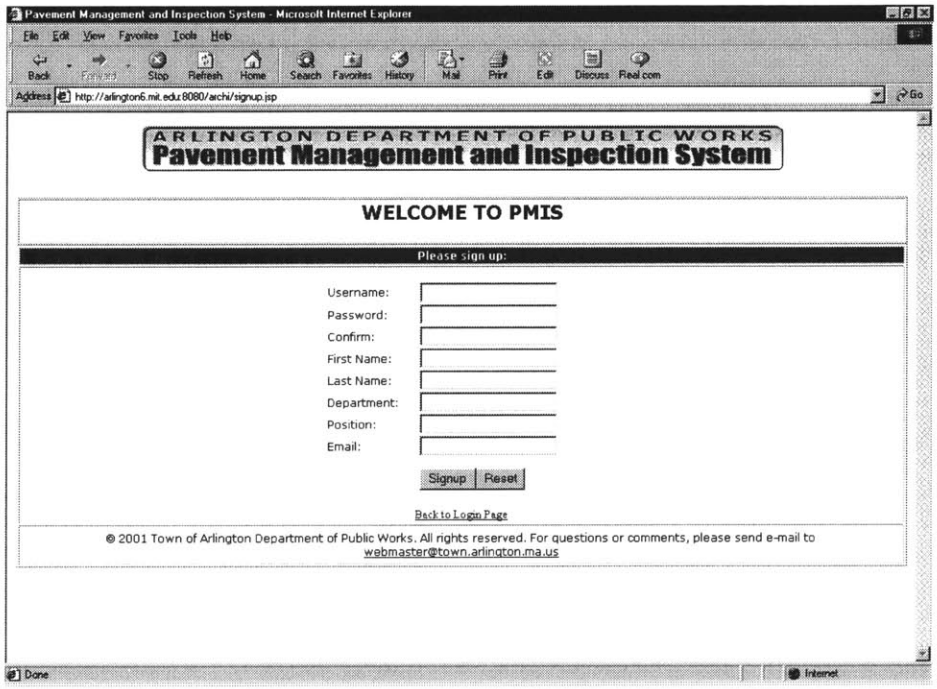


Figure 3-4: Register page for new account

3.6.2 PMIS Main Page

After the login process is successfully completed, the system brings the user to the PMIS main page, shown below in Figure 3-5. The user is allowed to navigate the system functions. These functions provided in the system are

- *Pavement Analysis*: manages the street maintenance plans (scenarios)
- *Inspection*: manages the street inspection files from the inspection process
- *Report*: views results, such as the maintenance plans, defect data, etc. in a user-friendly format
- *Permit System*: provides a link to Permit System homepage
- *Administration*: manages the database
- *FAQ*: provides help information

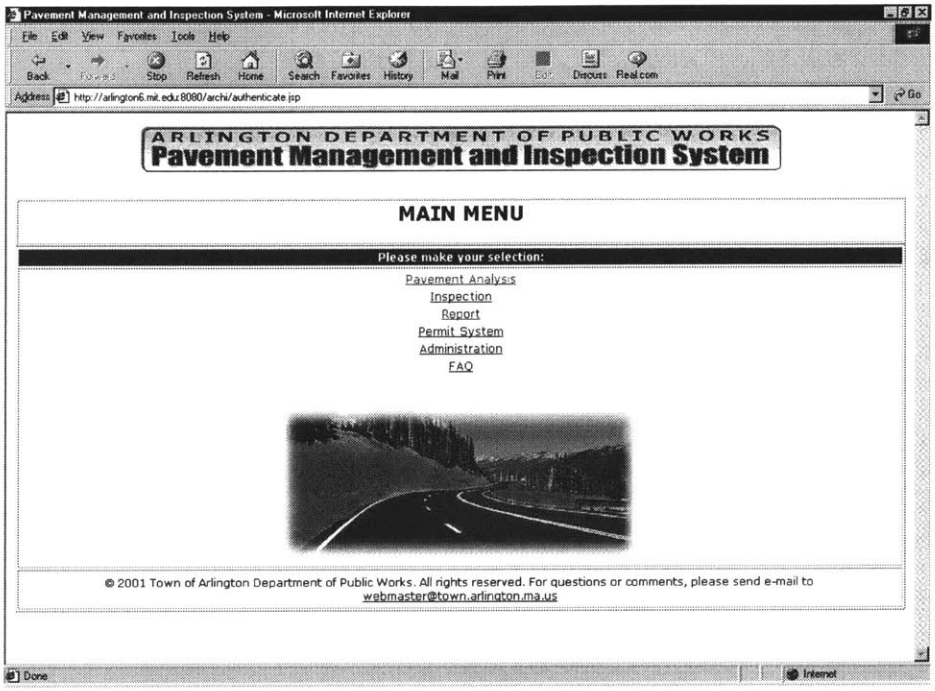


Figure 3-5: PMIS Main page

3.6.3 Pavement Analysis

Under the *Pavement Analysis* function, there are 3 sub functions, which are

- *Create Scenario*: allows the user to create new scenario in a particular year
- *Load Scenario*: allows the user to view the details of selected scenario
- *Compare Scenario*: allows the user to compare the scenarios in a specific year

The page for the pavement analysis is presented in Figure 3-6. The user needs to specify the year in which the desired scenario will be created (to create) or has been created (to load or compare). Refer to Evolution of Platform and User Interface in Infrastructure Management System with Case Study of Arlington Pavement Management System by Choi for more details on Pavement Analysis function.

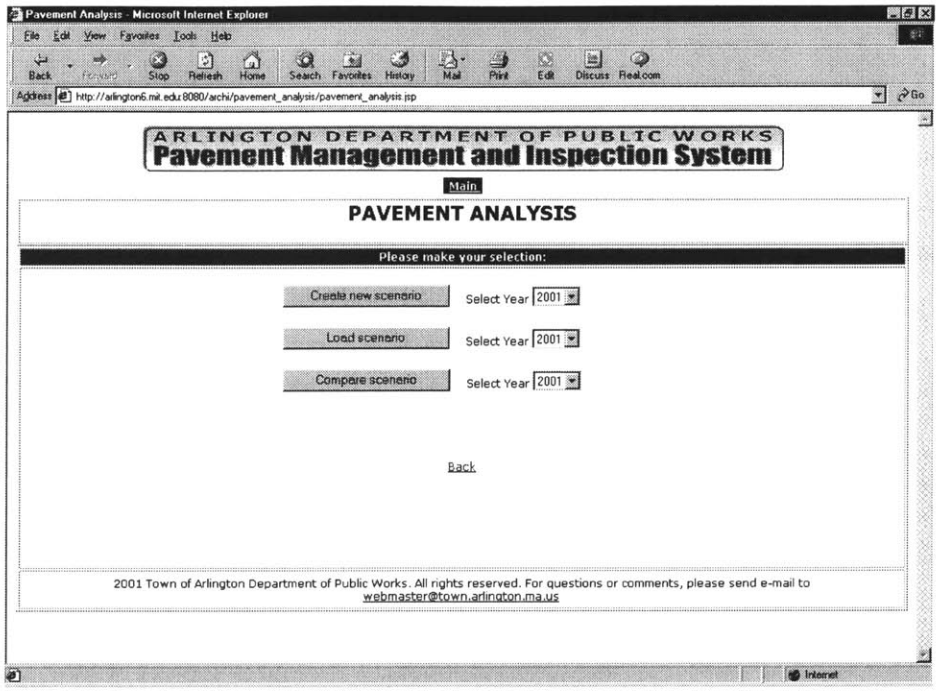


Figure 3-6: Pavement Analysis page

If the user chooses to create a new scenario, the name of the scenario needs to be entered, including its comment. On the other hand, if the user selects to load a scenario that already exists in the system, the system provides the list of scenario names for the user to choose from. After the user enters or selects the name, the system retrieves data from the database and presents it to the user.

Figure 3-7 and Figure 3-8 illustrate the results after user creates a new scenario, and loads an existing scenario, respectively.

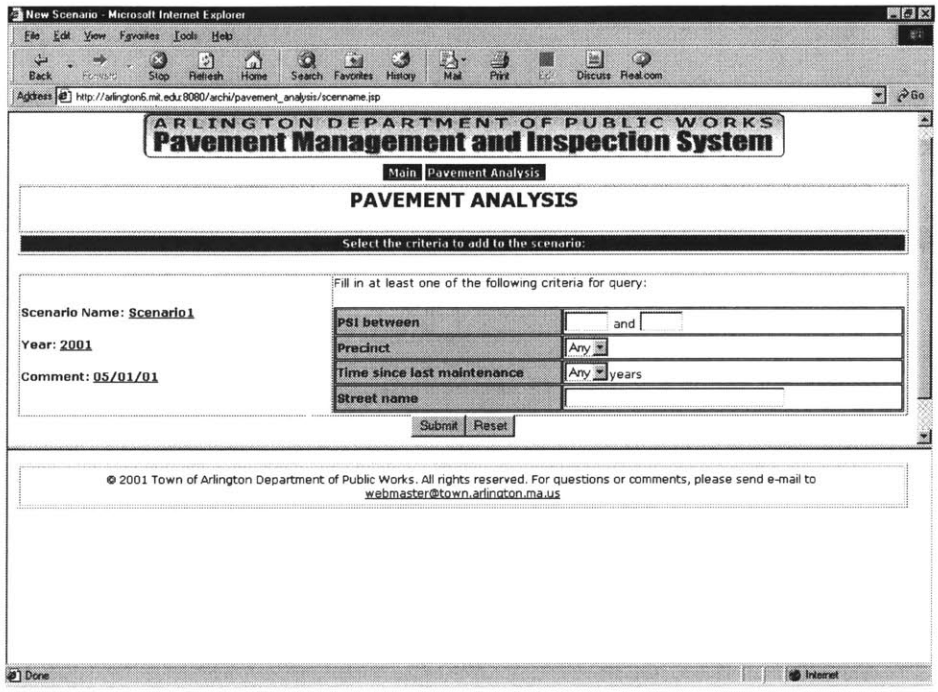


Figure 3-7: Create new Scenario

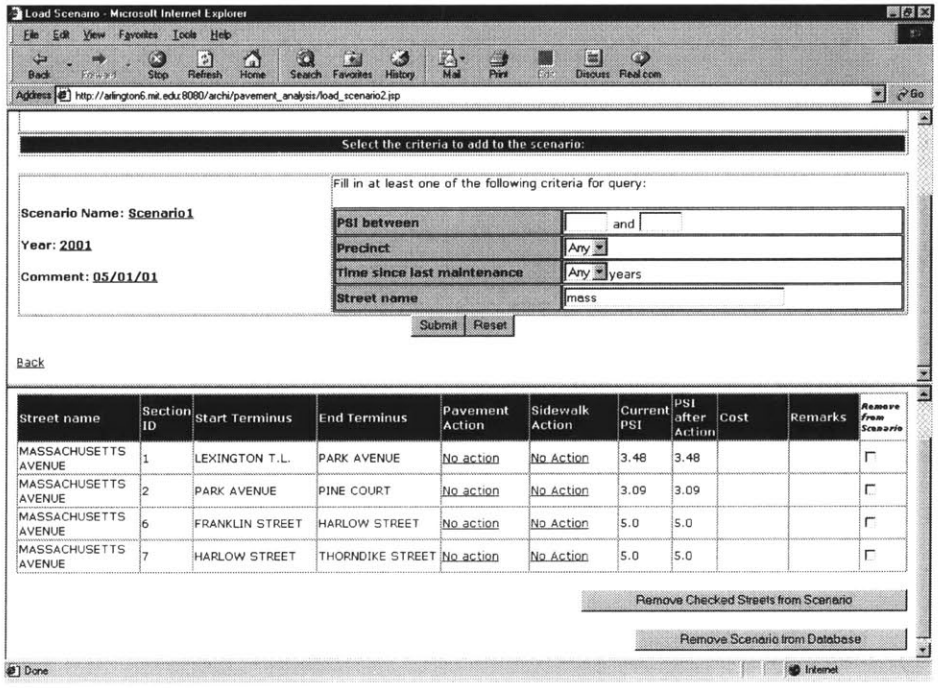


Figure 3-8: Load Scenario

If the user wants to add more streets to the scenario, he can select the criteria, which could be PSI, Precinct, Last maintenance date, and Street name. Then all street sections meeting the criteria will be appended to the list of the streets. The top part of the page in Figure 3-7 and Figure 3-8 is the area where the user can enter all the criteria to add streets. The user can also remove streets from the scenario or remove the scenario by checking the box provided at the bottom right of the page.

For each street section in the scenario, the user can select the pavement action, curb action, or sidewalk action to apply to it. The cost of maintenance for each action, as well as its benefit-cost ratio, is automatically estimated for the user to select the best maintenance action. The pavement action page is presented in Figure 3-9.

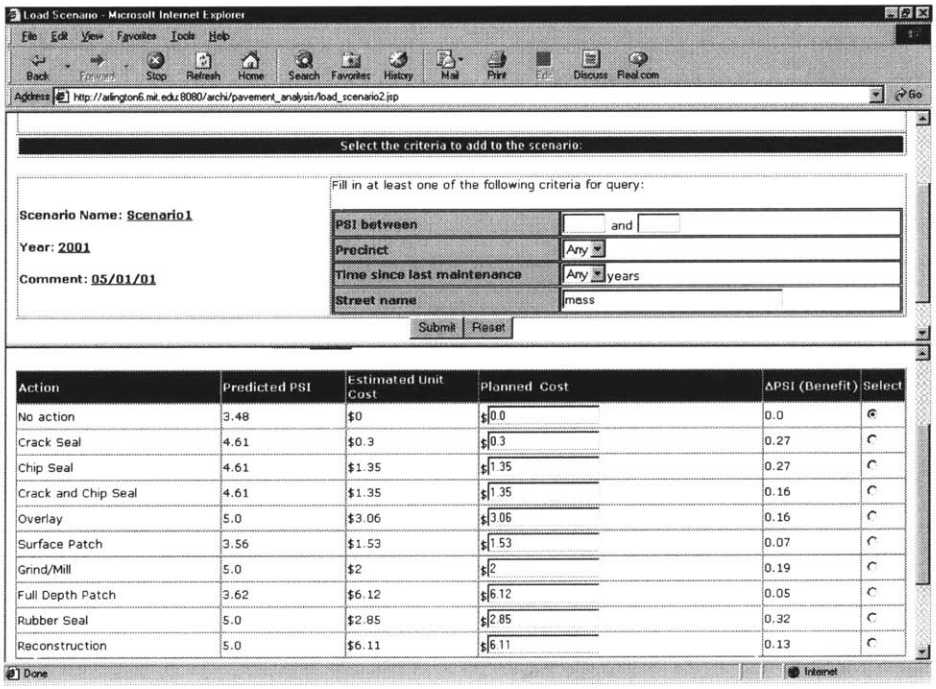


Figure 3-9: Pavement Action page

If the user chooses to compare scenarios, the list of all scenario names in that particular year will be displayed as in Figure 3-10. The user can choose to see more details of each scenario by checking the box provided for each scenario as in Figure 3-11. Also, after viewing the scenario details, the system provides a function to export the information to

MS Excel for further analysis by utilizing Excel functions, such as graphics or data analysis.

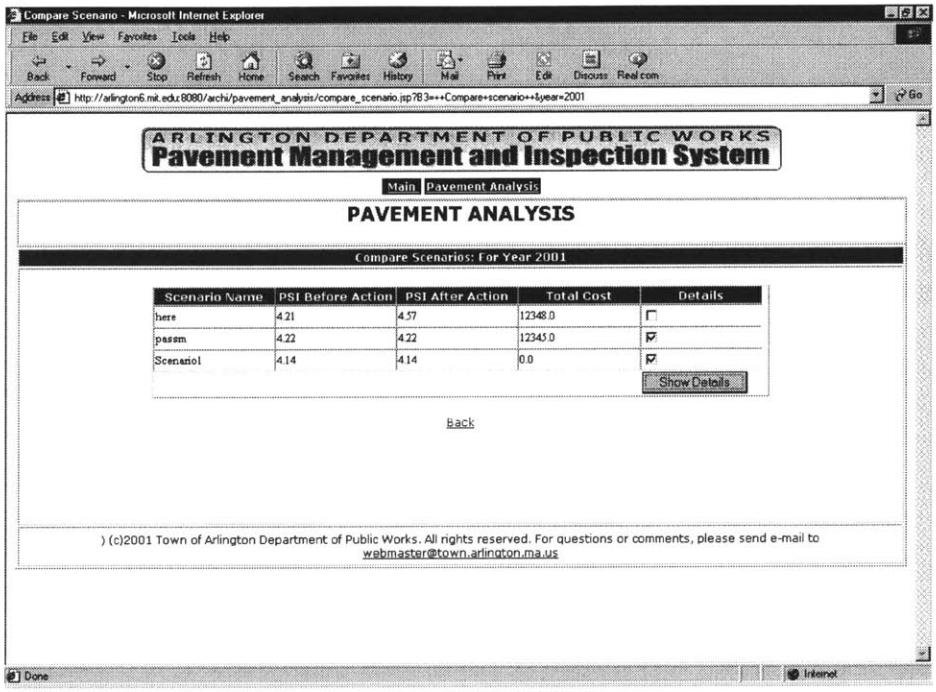
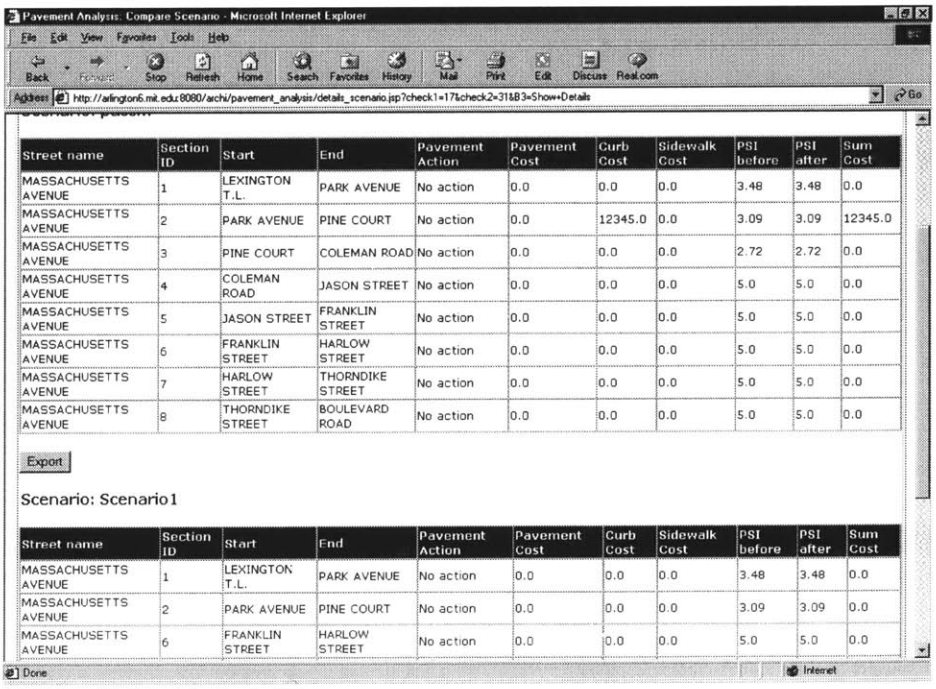


Figure 3-10: Compare Scenario page



Street name	Section ID	Start	End	Pavement Action	Pavement Cost	Curb Cost	Sidewalk Cost	PSI before	PSI after	Sum Cost
MASSACHUSETTS AVENUE	1	LEXINGTON T.L.	PARK AVENUE	No action	0.0	0.0	0.0	3.48	3.48	0.0
MASSACHUSETTS AVENUE	2	PARK AVENUE	PINE COURT	No action	0.0	12345.0	0.0	3.09	3.09	12345.0
MASSACHUSETTS AVENUE	3	PINE COURT	COLEMAN ROAD	No action	0.0	0.0	0.0	2.72	2.72	0.0
MASSACHUSETTS AVENUE	4	COLEMAN ROAD	JASON STREET	No action	0.0	0.0	0.0	5.0	5.0	0.0
MASSACHUSETTS AVENUE	5	JASON STREET	FRANKLIN STREET	No action	0.0	0.0	0.0	5.0	5.0	0.0
MASSACHUSETTS AVENUE	6	FRANKLIN STREET	HARLOW STREET	No action	0.0	0.0	0.0	5.0	5.0	0.0
MASSACHUSETTS AVENUE	7	HARLOW STREET	THORNDIKE STREET	No action	0.0	0.0	0.0	5.0	5.0	0.0
MASSACHUSETTS AVENUE	8	THORNDIKE STREET	BOULEVARD ROAD	No action	0.0	0.0	0.0	5.0	5.0	0.0

Street name	Section ID	Start	End	Pavement Action	Pavement Cost	Curb Cost	Sidewalk Cost	PSI before	PSI after	Sum Cost
MASSACHUSETTS AVENUE	1	LEXINGTON T.L.	PARK AVENUE	No action	0.0	0.0	0.0	3.48	3.48	0.0
MASSACHUSETTS AVENUE	2	PARK AVENUE	PINE COURT	No action	0.0	0.0	0.0	3.09	3.09	0.0
MASSACHUSETTS AVENUE	6	FRANKLIN STREET	HARLOW STREET	No action	0.0	0.0	0.0	5.0	5.0	0.0

Figure 3-11: Details of compared scenarios

3.6.4 Inspection

Under the *Inspection* function, there are 3 sub functions, which are

- *Upload Palm Data File*: uploads the inspection file from Palm to store in the system
- *Inspection Database Administration*: manages the inspection data in the database system
- *Palm Database Backup File*: saves the inspection files from the Web system to a user's own computer

The inspection page is presented in Figure 3-12. Refer to Evaluation of Infrastructure Monitoring System Using PDA and GPS Technologies by Cheung for more details on the PMIS Inspection function.

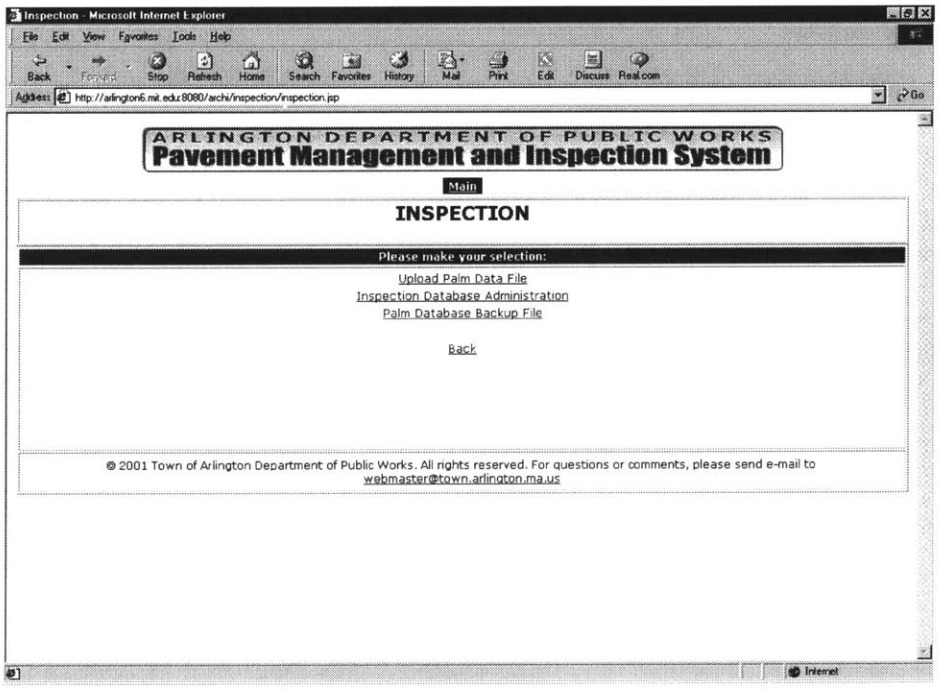


Figure 3-12: Inspection Page

3.6.5 Report

Under the *Report* function, there are 5 sub functions, which are

- *Distribution of PSI*: displays the numbers of streets that have their current PSI values corresponding to the ranges of PSI (0-2, 2-3, 3-4 and 4-5)
- *Distribution of Maintenance Date*: displays the numbers of streets that have been maintained within the particular year
- *History of Maintenance Costs*: displays average costs spent for the street maintenance action.
- *Summary of Defects*: displays the average numbers of each defect type from the last updated inspection or construction data in the database.
- *Summary of Maintenance Street*: displays details of each street that has had maintenance actions within the last year

The inspection page depicted in Figure 3-13 provides the links to generate each report for the user to choose. Again, the user is allowed to export these reports to MS Excel to perform further analysis. Figure 3-14 and Figure 3-15 illustrate the form of the reports that are generated by choosing *History of Maintenance Cost* and *Summary of Defect* respectively.

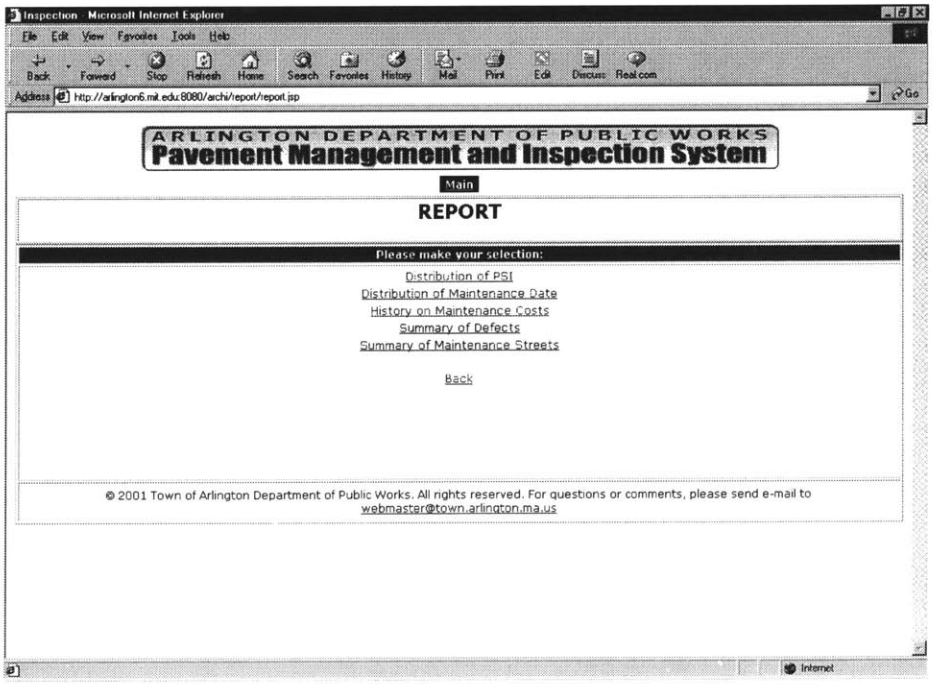


Figure 3-13: Report page

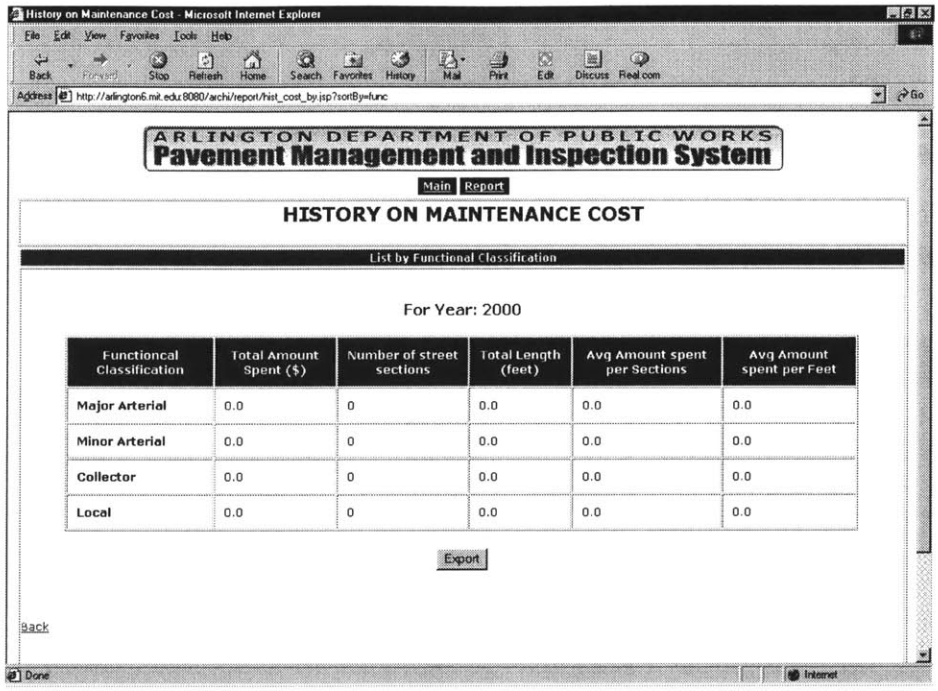


Figure 3-14: History of Maintenance Costs report

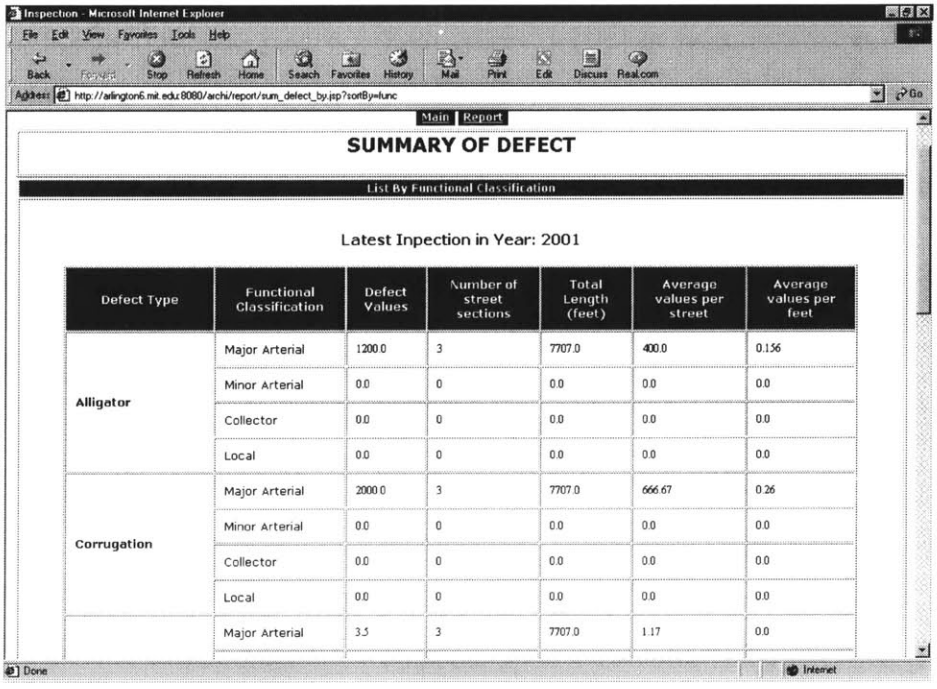


Figure 3-15: Summary of Defects report

3.6.6 Permit System

The *Permit System* function brings the user to the home page of the Permit System that allows users to access the street opening permit application form for Arlington. Figure 3-16 presents the home page of this system. Refer to Master of Engineering Project Report: Street Opening Permit System by Klimke, Qi, and Prasad for more details on the Permit System.

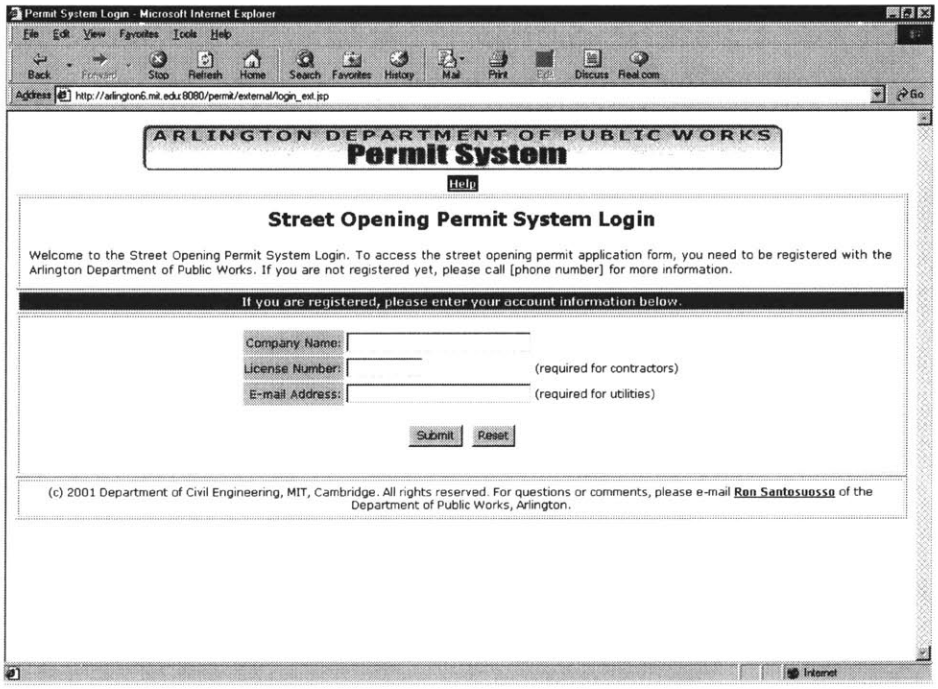


Figure 3-16: Permit System home page

3.6.7 Administration

Under the *Administration* function, there are 9 sub functions as shown in Figure 3-17. These functions allow the user to add or edit the records in the PMIS database. These functions are

- *Street data*: manages the street data table in database
- *Pavement action*: manages the pavement action table in database
- *Curb action*: manages the curb action table in database
- *Sidewalk action*: manages the sidewalk action table in database

- *Pavement Type*: manages the pavement type table, which describes the material type of the streets, the in database
- *Facility Type*: manages the facility type table, which contains the type of all streets, in database
- *Functional Classification*: manages the functional classification table, which is the level of use for the street, in database
- *Precinct*: manages the precinct table in database
- *User Information*: manages his profile (username, password, etc.) in the user table in database

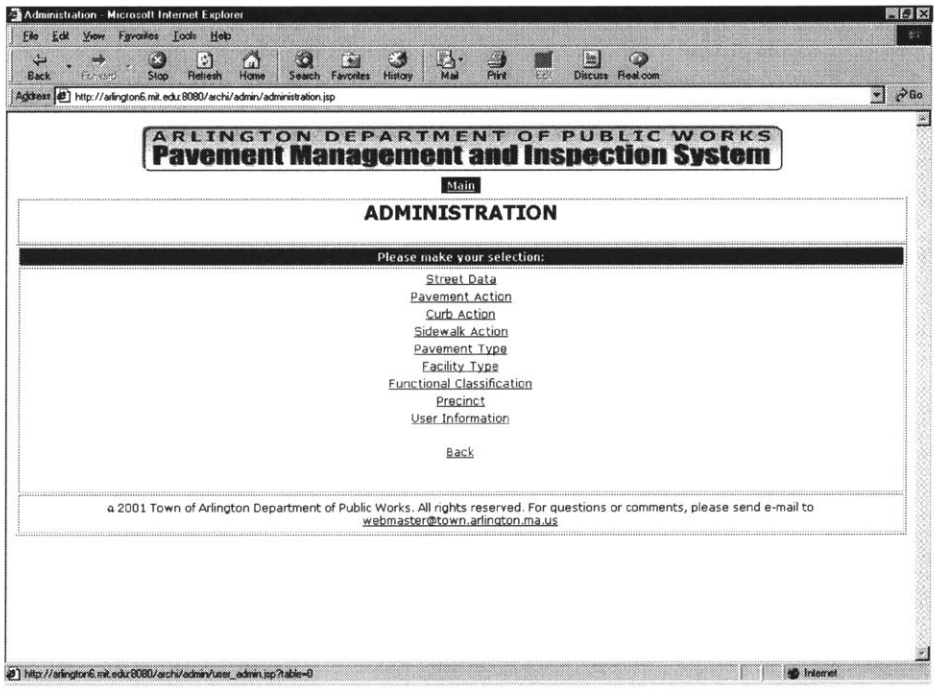
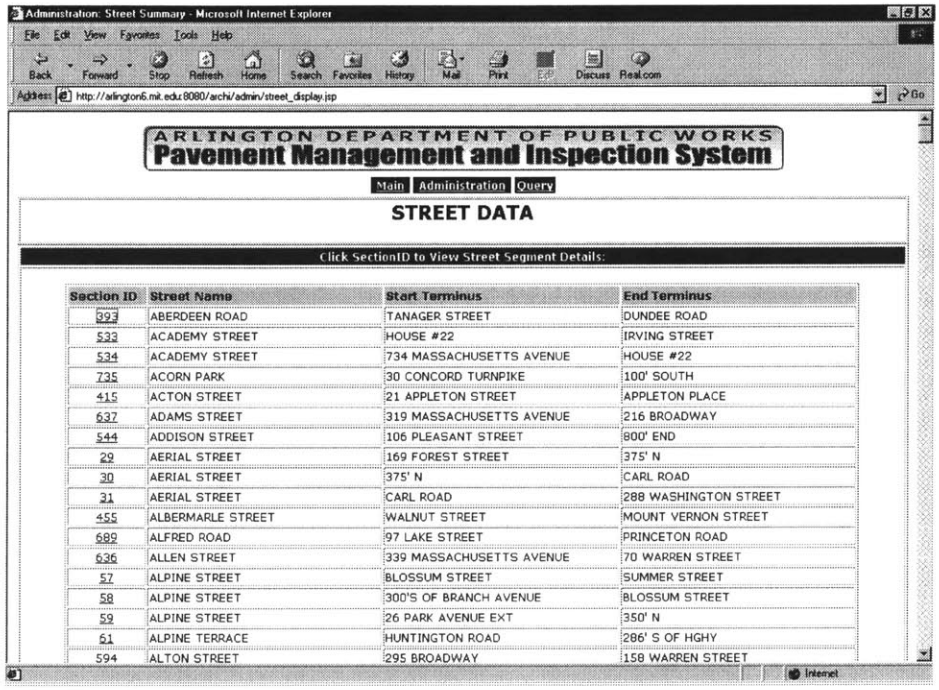


Figure 3-17:Administration page

Basically, when the user selects any link to manage a table in database, the system displays all the records in that selected table to the user. Then the user can choose to edit the existing records or add a new record to the database. Refer to Data Modeling in a Pavement Management System by Yim for details on PMIS database modeling.

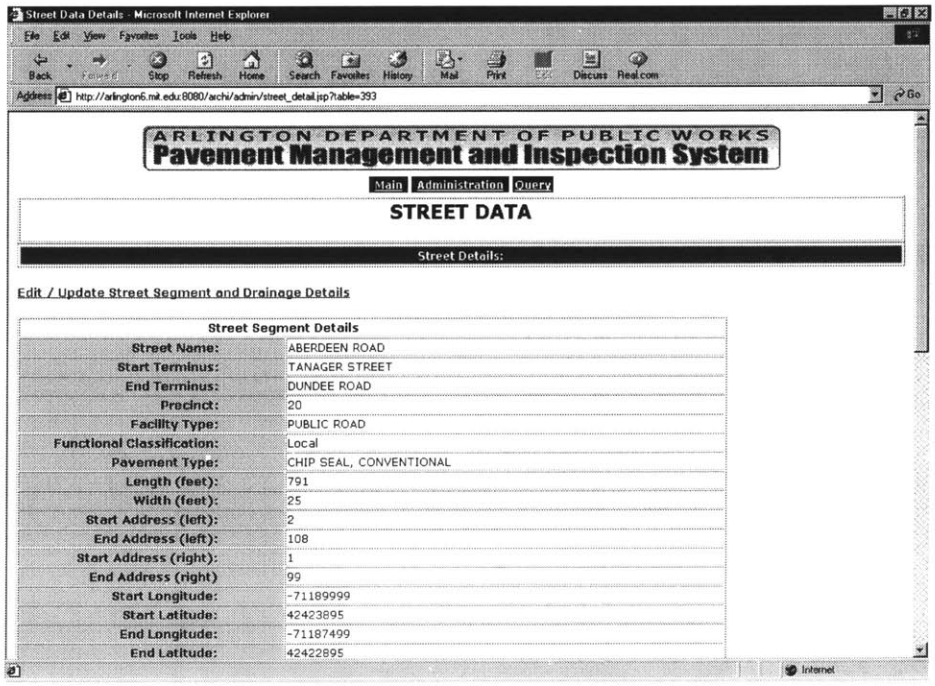
Figure 3-18 illustrates one example of an administration function, *Street Data* that displays the list of all street sections in the database. If the user wants to view the details

of any particular street, he can do it by clicking on the section number of that particular street. All the information for that street will be provided to the user as shown in Figure 3-19.



Section ID	Street Name	Start Terminus	End Terminus
393	ABERDEEN ROAD	TANAGER STREET	DUNDEE ROAD
533	ACADEMY STREET	HOUSE #22	IRVING STREET
534	ACADEMY STREET	734 MASSACHUSETTS AVENUE	HOUSE #22
735	ACORN PARK	30 CONCORD TURNPIKE	100' SOUTH
415	ACTON STREET	21 APPLETON STREET	APPLETON PLACE
637	ADAMS STREET	319 MASSACHUSETTS AVENUE	216 BROADWAY
544	ADDISON STREET	106 PLEASANT STREET	600' END
29	AERIAL STREET	169 FOREST STREET	375' N
30	AERIAL STREET	375' N	CARL ROAD
31	AERIAL STREET	CARL ROAD	288 WASHINGTON STREET
455	ALBERMARLE STREET	WALNUT STREET	MOUNT VERNON STREET
689	ALFRED ROAD	97 LAKE STREET	PRINCETON ROAD
636	ALLEN STREET	339 MASSACHUSETTS AVENUE	70 WARREN STREET
57	ALPINE STREET	BLOSSUM STREET	SUMMER STREET
58	ALPINE STREET	300'S OF BRANCH AVENUE	BLOSSUM STREET
59	ALPINE STREET	26 PARK AVENUE EXT	350' N
61	ALPINE TERRACE	HUNTINGTON ROAD	286' S OF HGHY
594	ALTON STREET	295 BROADWAY	158 WARREN STREET

Figure 3-18: Display list of all street sections in database



Street Segment Details	
Street Name:	ABERDEEN ROAD
Start Terminus:	TANAGER STREET
End Terminus:	DUNDEE ROAD
Precinct:	20
Facility Type:	PUBLIC ROAD
Functional Classification:	Local
Pavement Type:	CHIP SEAL, CONVENTIONAL
Length (feet):	791
Width (feet):	25
Start Address (left):	2
End Address (left):	108
Start Address (right):	1
End Address (right)	99
Start Longitude:	-71189999
Start Latitude:	42423895
End Longitude:	-71187499
End Latitude:	42422895

Figure 3-19: Display more details for the particular street section

The user is allowed to edit or change existing records in the database by selecting the provided link. Figure 3-20 illustrates the page to modify the data. The user can enter the information for that street in the provided fields. Moreover, for the fields that have specific data such as Precinct, Facility type, Functional classification, or Pavement type, he can select from the drop-down list that system provides.

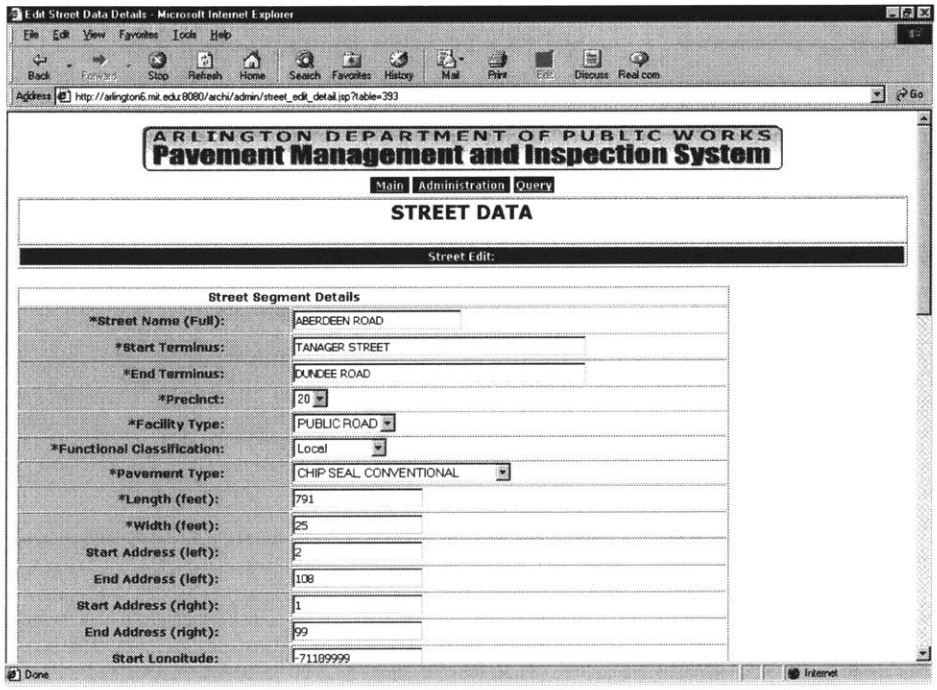


Figure 3-20: Edit the street data

From the introduction of the PMIS system described above, it can be seen that the user has much more convenience than the old IMS DOS-based system. The user can access the system from remote locations. Furthermore, the changes in the user interface improve the usability of the system substantially. For instance, the user does not need to remember the street section number or all the number that represented types, as he had to do with the old system. Instead, he can just specify a street name or select from the provided list as shown in Figure 3-20. These changes are considered to be the most important improvements of the new system, as they are the primary needs of the client the Arlington DPW.

3.7 PMIS Use Case and Sequence Diagrams

Use cases and sequence diagrams are two of the nine diagrams types in the Unified Modeling Language (UML), which were originally created in early of 1990s. These two diagrams are very popular aids in designing and developing a software product. The use case diagrams are informal, very easy to use and understand for both technical as well as non-technical people. Also, the sequence diagrams emphasize the order in which things occur. In this section, the use case diagrams and the sequence diagrams of the PMIS project are described in detail.

3.7.1 Use Case Diagrams

Use case diagrams describe the functionality of a system and users of the system. These diagrams consist of two elements. The first element is *Actors*, which represent users of a system, including human users and other systems. The other element is *Use cases*, which represent functionality or services provided to actors.

3.7.2 Sequence Diagrams

Sequence diagrams are models that describe how groups of objects collaborate in some behavior. The objective of the sequence diagrams is to explain the interactions among classes. The diagrams focus on classes and the messages they exchange to achieve the desired behavior. Typically, the diagrams show a number of objects and the messages that are passed between these objects within the use cases.

3.7.3 PMIS Actors

3.7.3.1 Pavement Manager

The pavement managers can be more than one person who uses the PMIS to manage the pavement maintenance plans. They must be the engineers at the DPW in Arlington in order to access and work in this system.

3.7.3.2 Inspector

The inspectors are engineers who inspect streets in Arlington and collect defect data by using Palm and GPS equipment.

3.7.4 PMIS Use Case Diagrams

Use case diagrams of the entire PMIS system including the web-based application and the Palm-application can be generally illustrated in Figure 3-21.

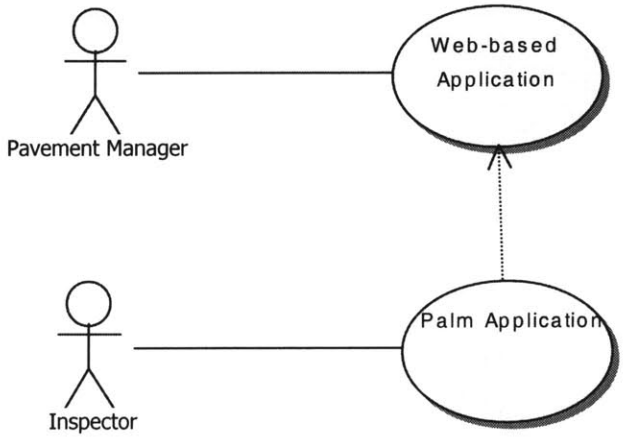


Figure 3-21: Use case diagrams for the PMIS system

Note that in this thesis, only the use cases for the Pavement manager (the Web-based application) will be discussed.

3.7.5 Web-based Application Use Case Diagrams

Use case diagrams for the pavement manager are presented in Figure 3-22. The paragraphs following explain each use case, along with its text descriptions, according to the functions provided in the PMIS system in details.

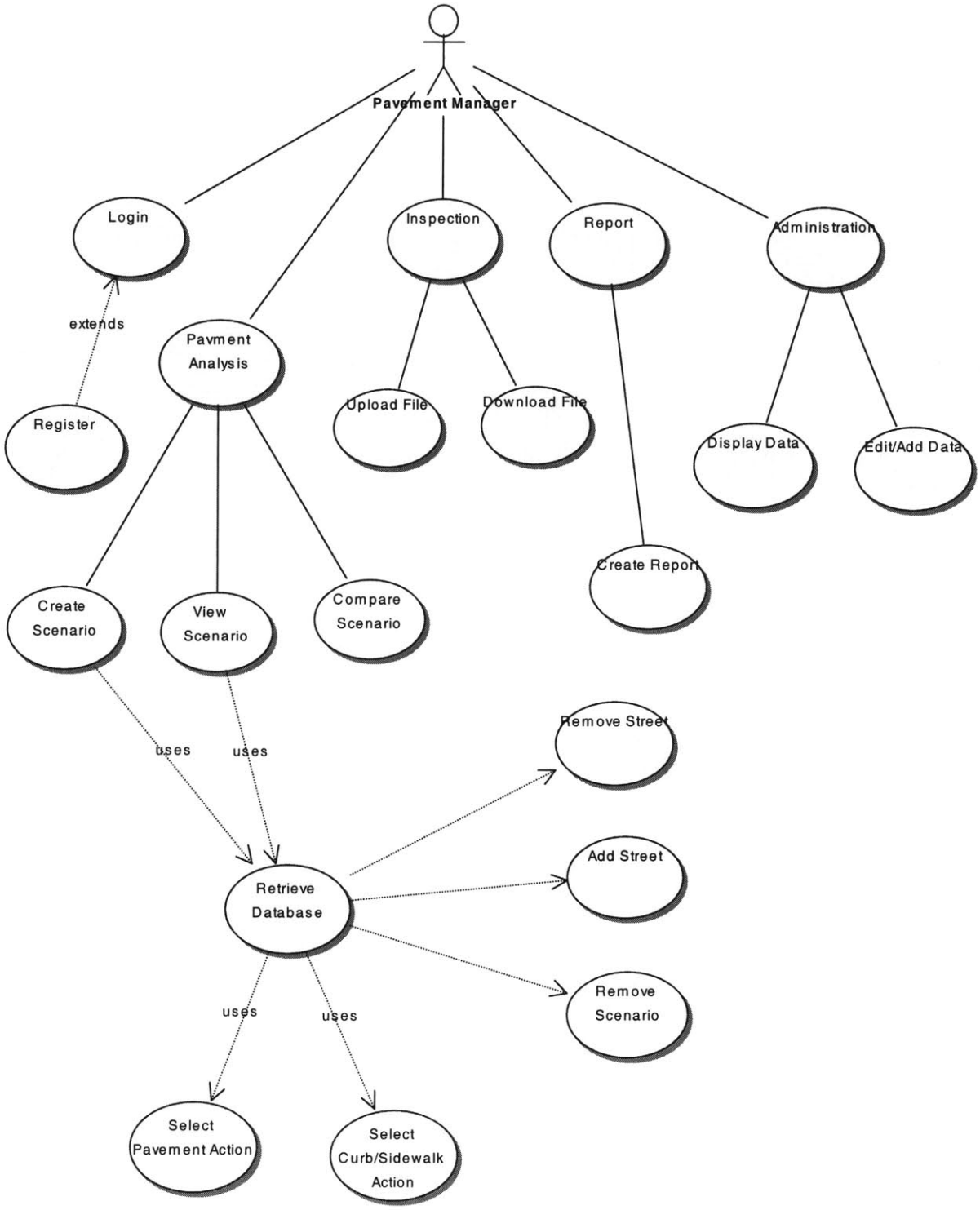


Figure 3-22: Use case diagrams for Web-based Application

3.7.5.1 Login page use cases

The system starts with this page to verify the user before starting the system by using these use cases.

Login

This use case is for the existing user to log into the system

- The user enters his login name and password
- The system checks if the login name and password are correct and exist in the database
- If yes, the system displays the main functions
- If not, the system displays the error page and let the user enters data again.

Precondition

- The user must have the login name with the corresponding password for entering the system

Register

This use case is for a new user, who needs to create a new account in order to log into the system. The new user can register for the account by selecting a provided link

- The new user enters his information including username, password, first name, last name, department, position, and e-mail address
- The system checks if the requested username is unique in the database to create the new account
- If yes, the system creates a user profile for the user and stores his information into the database
- If not, the system allows the user to register again

Post condition

- After registering, the system assigns the unique user ID for the successful registered user. Then, the user can log into the system immediately

3.7.5.2 Pavement Analysis use cases

This set of use cases allows the user to manage the street maintenance plans.

Create scenario

This use case lets the user to create a new scenario in the database system

- The user selects the year of the scenario that he wants to create
- The user fills in the following information to indicate a scenario
    1. Scenario Name
    2. Comments (optional)
- The system verifies the uniqueness of the scenario name in the database
- If the name exists in the database, the system allows the user to enter the new name
- If the name is unique, the system stores the scenario name, the selected year, comments and the username into the database
- The system retrieves and displays a set of streets in that scenario in database by using a *Retrieve database* use case. The use case will be described later.

View scenario

This use case lets the user choose an existing scenario in the database system to be viewed or edited.

- The user selects a name and year whose scenario he wants to view or edit
- The system retrieves and displays a set of streets in that scenario from the database by using *Retrieve database* use case.

Retrieve database

This use case retrieves data from the database and displays them to the user

- Given the specific year and the scenario name, the system retrieves the data from the database
- System returns the following information to the user

1. Streets information which includes name, section ID, start and end terminus)
2. The selected pavement action of each street section
3. The selected sidewalk action of each street section
4. The condition of each street, in terms of PSI values (including the condition before and after applying the pavement action)
5. The cost of action of each street section
6. Remarks

Precondition

- Inspection data must exist for each street to perform the PSI calculation
- The PSI calculation method must exist
- The cost model parameters must exist

Add street

This use case lets the user add streets and the corresponding data into the scenario by querying from the given criteria.

- User fills in one or more of the criteria which include PSI value, Precinct, Time since last maintenance, and Street name
- Given the above criteria, the system queries for the data that corresponding to the criteria and displays them to the user
- The system adds that data, which resulted from querying, to that particular scenario in the database

Precondition

- The scenario must exist in the system in order to add the new set of streets

Remove street

This use case lets the user to remove streets in the specific scenario from the database. User can select to remove more than one street at a time.

- The user chooses the streets that need to be removed from the scenario

- The system deletes the selected street section from the scenario in the database

Post condition

- The data about the maintenance actions (pavement, sidewalk and curb action) for that street is removed automatically when the street is removed from the scenario

Remove scenario

This use case allows the user to remove the existing scenario from the database.

- The user can select the scenario to be removed by clicking the provided button
- The system delete the scenario from the database

Post condition

- The data about the streets consisting in the deleted scenario including all maintenance actions for that street is removed automatically when the scenario is removed.

Select pavement action

This use case lets the user assign the pavement maintenance action to the street. Within this use case, there is a sub use case, which is the *Retrieve street details*. It will be described later.

- The user selects the street that he wants to add or edit the pavement maintenance action from the list of the streets within the specific scenario
- The system displays the information of the selected streets. This information includes
    1. Street name
    2. Start and end terminus
    3. Current street condition
    4. Estimated street condition in selected year
- The user selects the pavement action to apply to the particular section

- The user fills in the estimated cost of the selected action if the estimated cost from the system is not applicable
- The user can choose to add or discard the selected action to the street
- The system updates the pavement action to the street section in the database
- The system brings the user back to the list of the streets to repeat adding the action to the next street

Precondition

- The PSI after applying the action, estimated cost, and benefit-cost ratio of each maintenance action is calculated and displayed in order to help the user making the decision of selecting the action
- The street section ID must exits and is assigned for each street section

Select curb/sidewalk action

This use case lets the user assign the curb and sidewalk maintenance action to the streets. Within this use case, there is a sub use case, which is the *Retrieve street details.* It will be described later.

- The user selects the street for which he needs to add or edit the maintenance action from the list of the streets in the scenario
- The user selects the desired action by checking the check box displayed corresponding to each action. It can be more than one box for the curb action and only one box for the sidewalk action
- The user fills in the quantity of curb and sidewalk that need to be maintained in feet
- The system generates the estimated cost for each action automatically
- The user calculates the subtotal cost by clicking the provided button
- The user decides to add or discard the selected action to the street
- The system updates these selected actions in the database
- The system brings the user back to the list of the streets to repeat adding the action to the next street

Retrieve street details

This use case is a sub use case in the *Select pavement action* and *Select curb/sidewalk action* use case. It is to retrieve the detail of the selected street from the database and displays it to the user.

- Given the specific street section ID, system retrieves the its data from the database
- System returns information and shows it to the user

Compare scenario

This use case lets the user compare scenarios in a specific year to assist the user in determining the most effective scenario.

- The user selects a year for the scenarios to be compared from a provided list
- The system returns the following information
    1. Scenario name
    2. Average current PSI
    3. Average improved PSI after applied maintenance actions
    4. Total cost
- The user selects one or more scenarios to view more details by checking the box and submits the request
- The system shows the details of each scenario. This information includes
    1. Street Information (including name, ID, start and end terminus)
    2. Pavement action and its corresponding cost
    3. Curb action and its corresponding cost
    4. Sidewalk action and its corresponding cost
    5. Current PSI and PSI after action
    6. Total cost
- User can export the information to Microsoft Excel

Precondition

- The system displays only the scenarios that have not finished construction yet

3.7.5.3 Inspection use cases

This set of use cases allows the user or inspector to upload inspection files from the computer to the database system, manage the inspection files in the system, as well as download the existing files in the system back to a user's own computer. The defect data are very important to PMIS because they are involved in calculating the street conditions. The following sections describe the use cases of this function.

Upload file

This use case lets the user upload the inspection file to store in the system

- To save the inspection file in the system, the user can either type in the name of the file or browse through the file from a pop-up window that the system automatically generates
- The system accepts the file and displays its name, type and size
- The user can select to edit, finish, or cancel the file
- If the user selects to edit, system displays the defect data of each street containing in the file. The user can also remove any record before it will be saved into the database.
- If the user selects to finish, the system saves the defect data in that file directly into the database
- If the user selects to cancel, the system discards the file and not save it into the database.

Download file

This use case allows the user to download the inspection files that have been previously saved in the system to the user's computer. The user can also delete any inspection file existing in the system.

- The system generates the list of existing files

- To download the inspection file and save to the computer, the user selects the file name. The system automatically brings up the window to let the user save the file
- To delete the inspection file from the system, the user selects the files by checking the check boxes and press the provided button.

3.7.5.4 Report use cases

This function allows the user to view results in a user-friendly format. In addition, it allows the user to export these results to Microsoft Excel. There are 5 categories of report that user can generate from this function, as stated in Section 3.6.5. However, the use cases are the same for each report and can be described as below.

Create Report

- The user selects the type of report to be created
- The system connects to the database and retrieves the data
- The system displays the data to the user
- The user selects to export these data to Microsoft Excel

3.7.5.5 Administration use cases

This function allows the user to edit, update or delete records from the tables that exist in the database. For example, the Street data table, Pavement Action table, Curb Action table, Sidewalk Action table, or the User profile table can be administered.

The use cases of this function are very similar, so only the *Street Data* is presented, as it is the most complex feature.

Display Data

This use case lets user view the data of all street sections existing in the database, including the Construction History data and Drainage data of each street

- The system retrieves data from the database and displays all the section IDs, including the Street name, and the Start and End terminus. This data is in alphabetical order by Street name
- The user selects the street section ID in order to view more details
- The system retrieves data from the database and displays it to the user. These data include the data from the Street Data table, Construction History table, and Drainage table

Edit and Add Data

This use case allows the user to edit the data of street sections existing in the database or add a new street section into the database

- The user makes a change to or adds to the data
- The user submits information
- The system identifies whether this section ID exists in the database
- If yes, the system updates the data with that corresponding section ID
- If not, the system creates a new section ID and add the new record to the database

3.7.6 Web-based Application Sequence Diagrams

The figures below elaborate the sequence diagrams of the PMIS project. Only the diagrams of the *Login Page* function and the *Pavement Analysis* function are shown according to the use cases described in the section above, as they are important functions to the project and have interesting characteristics to study.

3.7.6.1 Login page sequence diagrams

Login

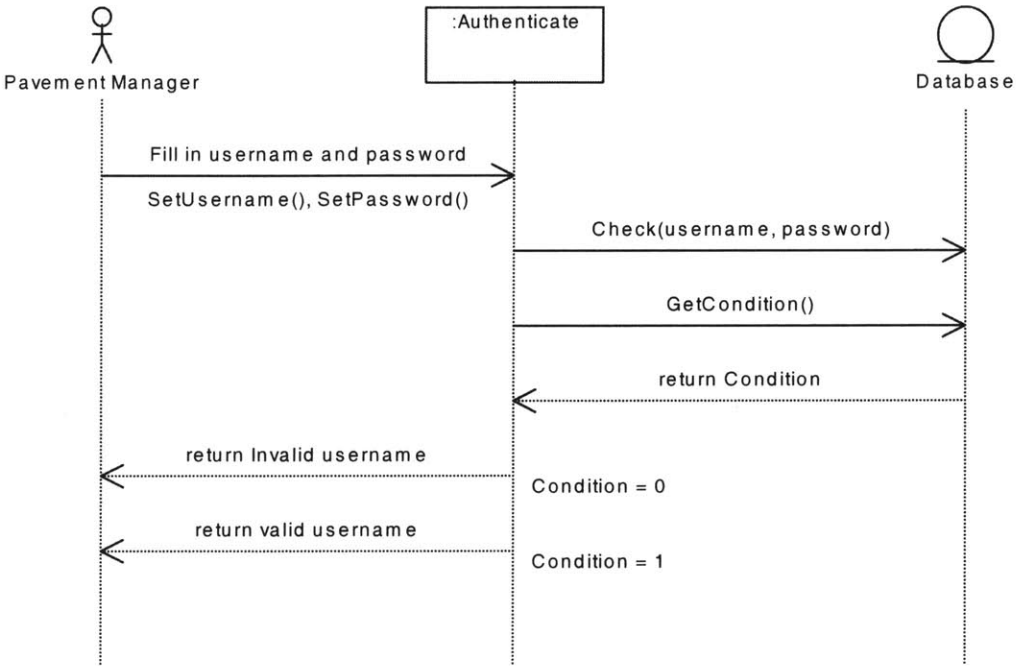


Figure 3-23: Login Sequence diagrams

Register

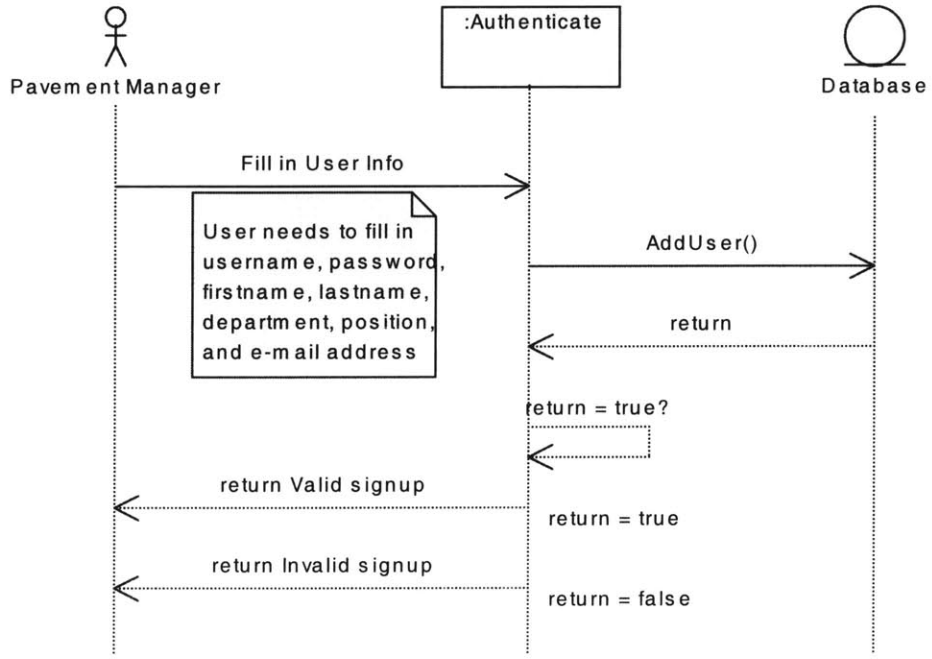


Figure 3-24: Register Sequence diagrams

3.7.6.2 Pavement Analysis sequence diagrams

Create scenario

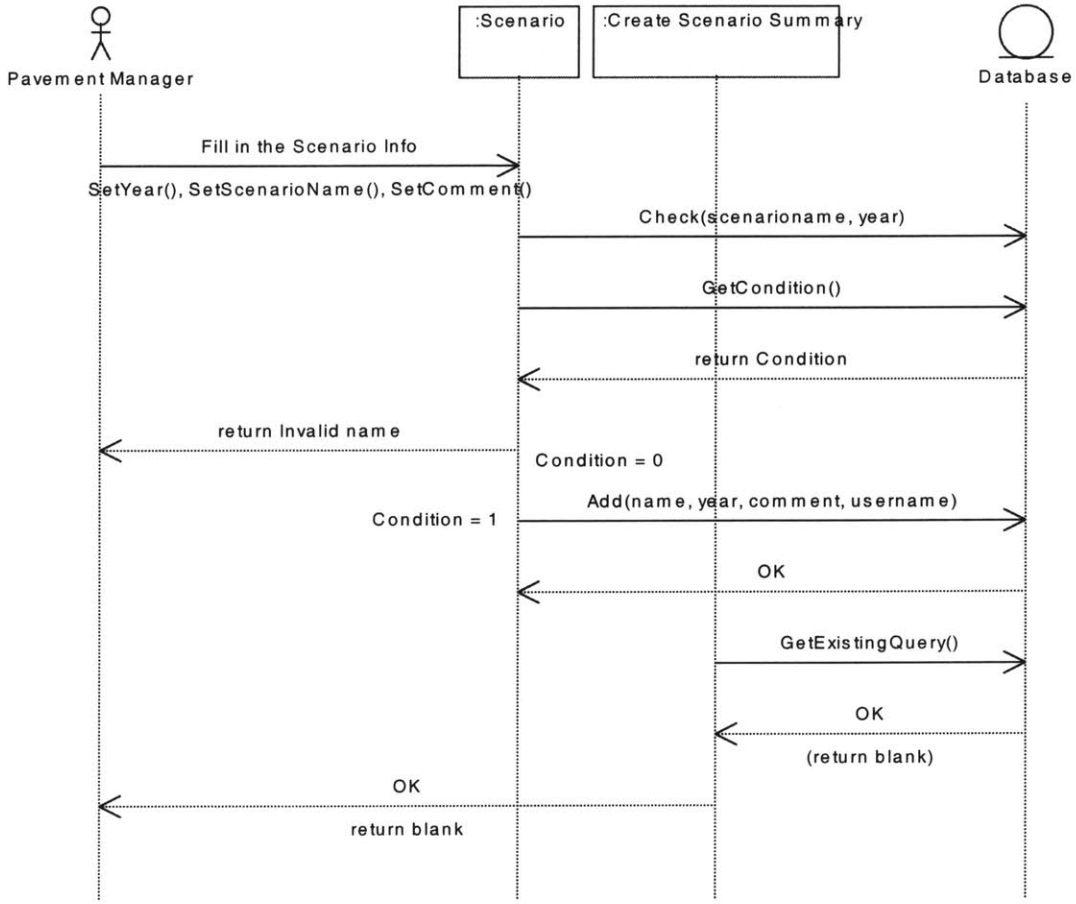


Figure 3-25: Create Scenario Sequence diagrams

View scenario

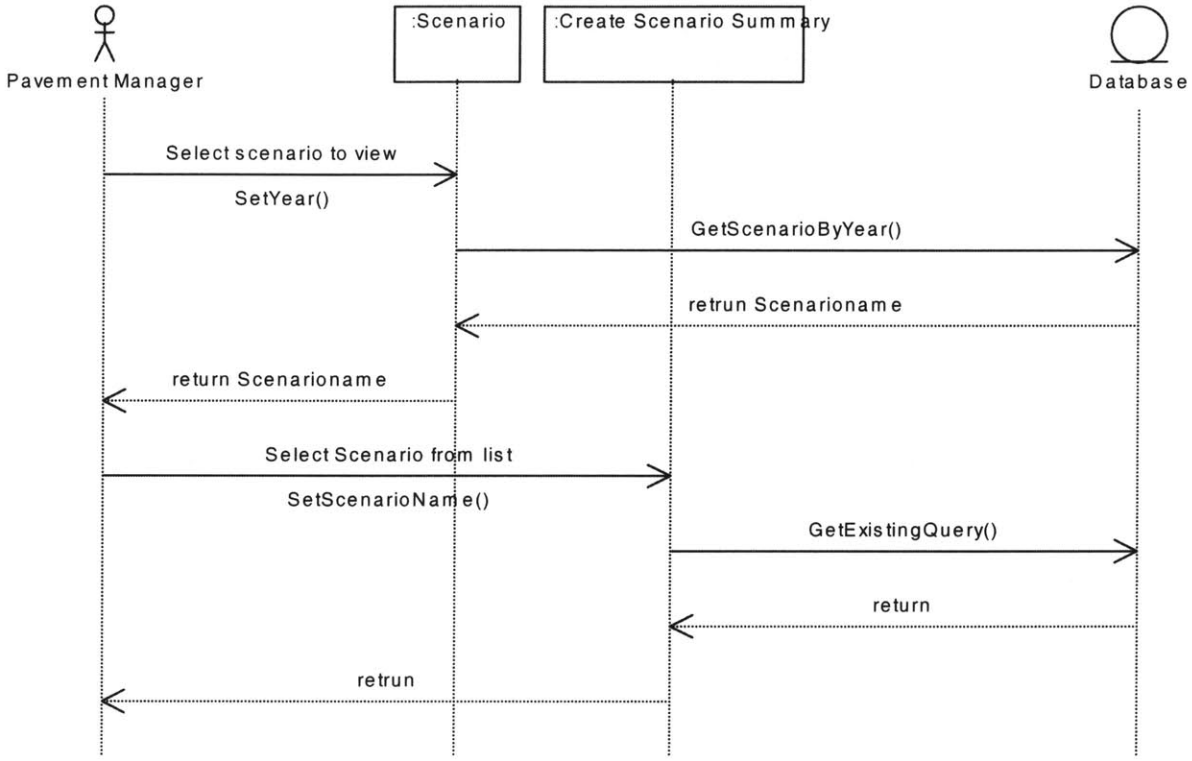


Figure 3-26: View Scenario Sequence diagrams

Compare scenario

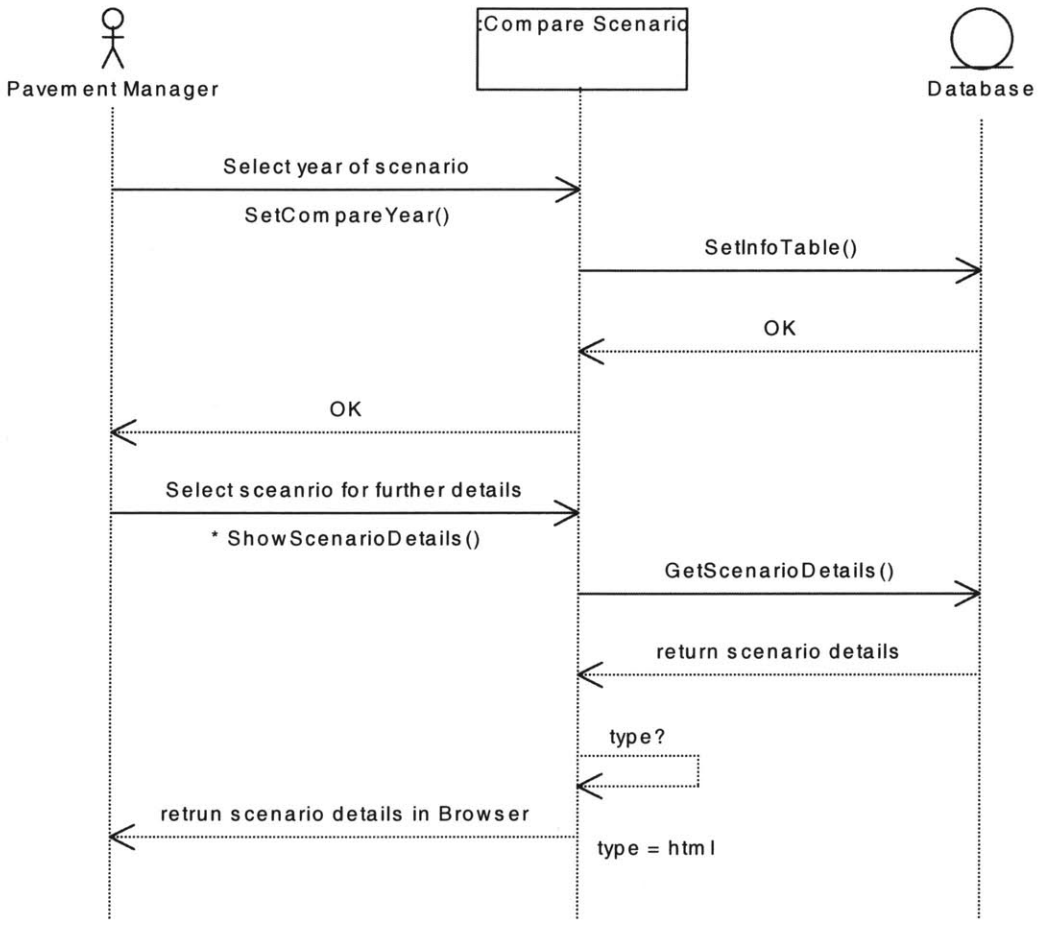


Figure 3-27: Compare Sequence diagrams

Add street

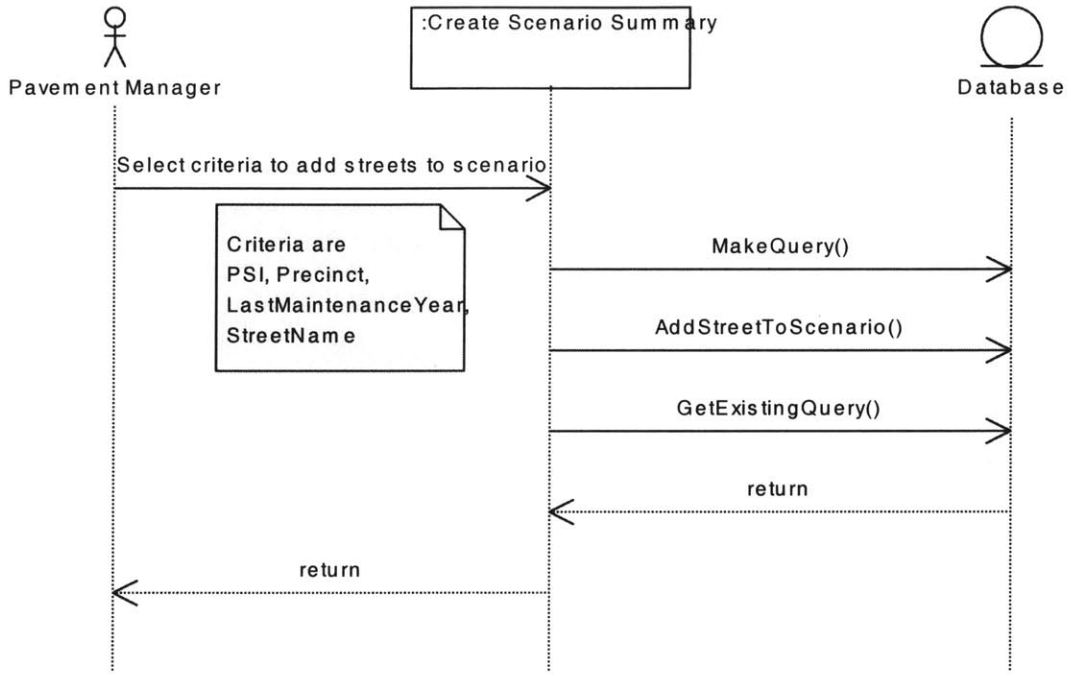


Figure 3-28: Add Street Sequence diagrams

Remove street

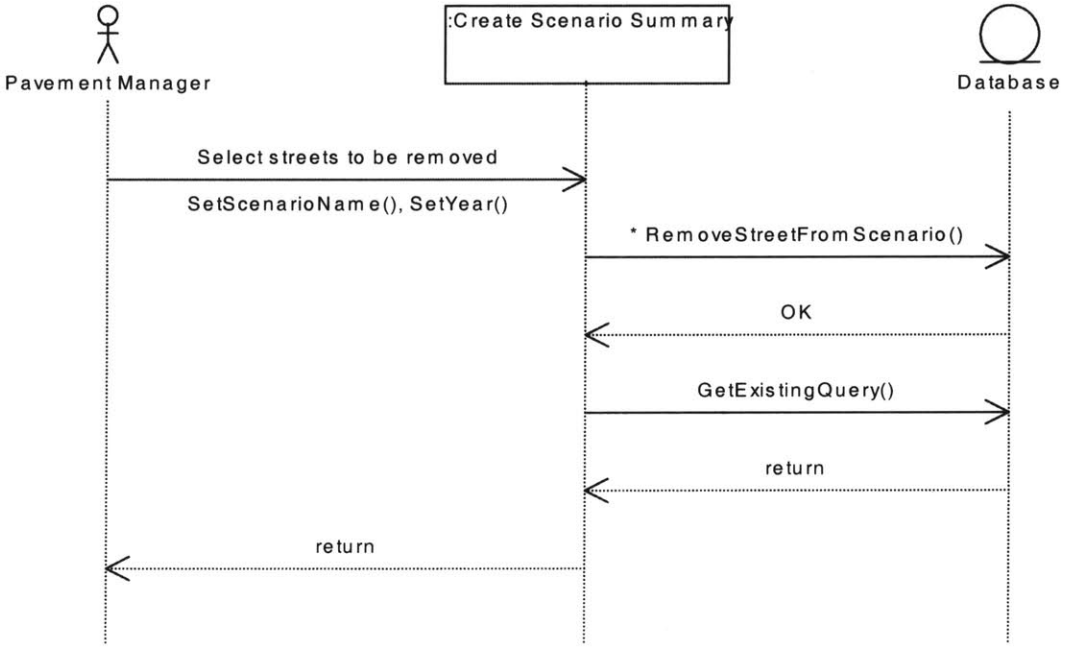


Figure 3-29: Remove Street Sequence diagrams

Remove scenario

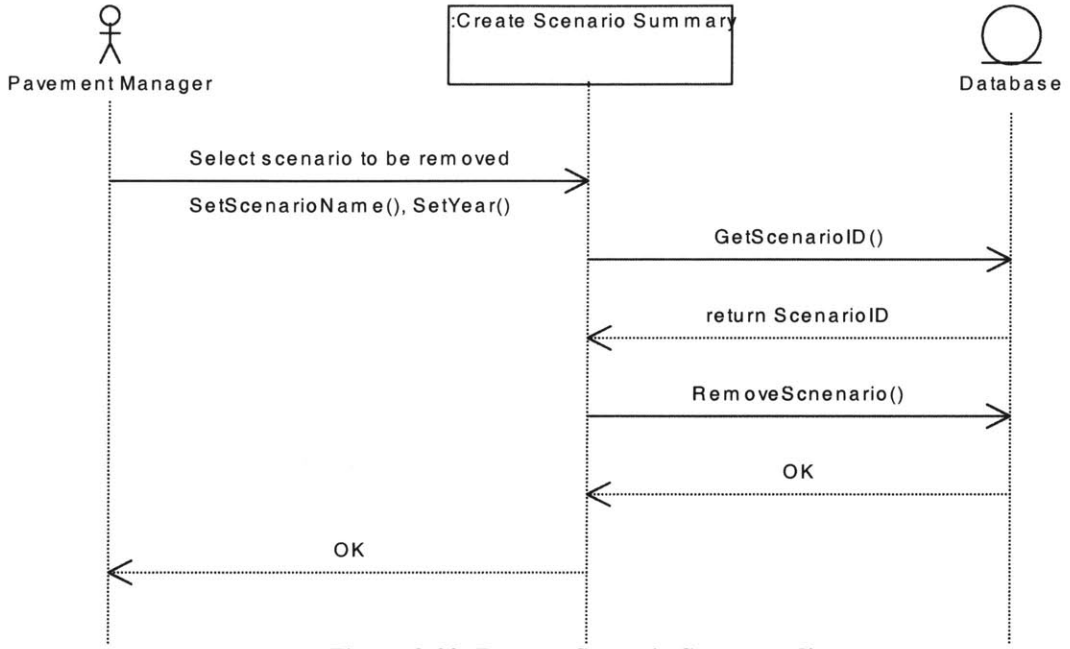


Figure 3-30: Remove Scenario Sequence diagrams

Select Pavement Action

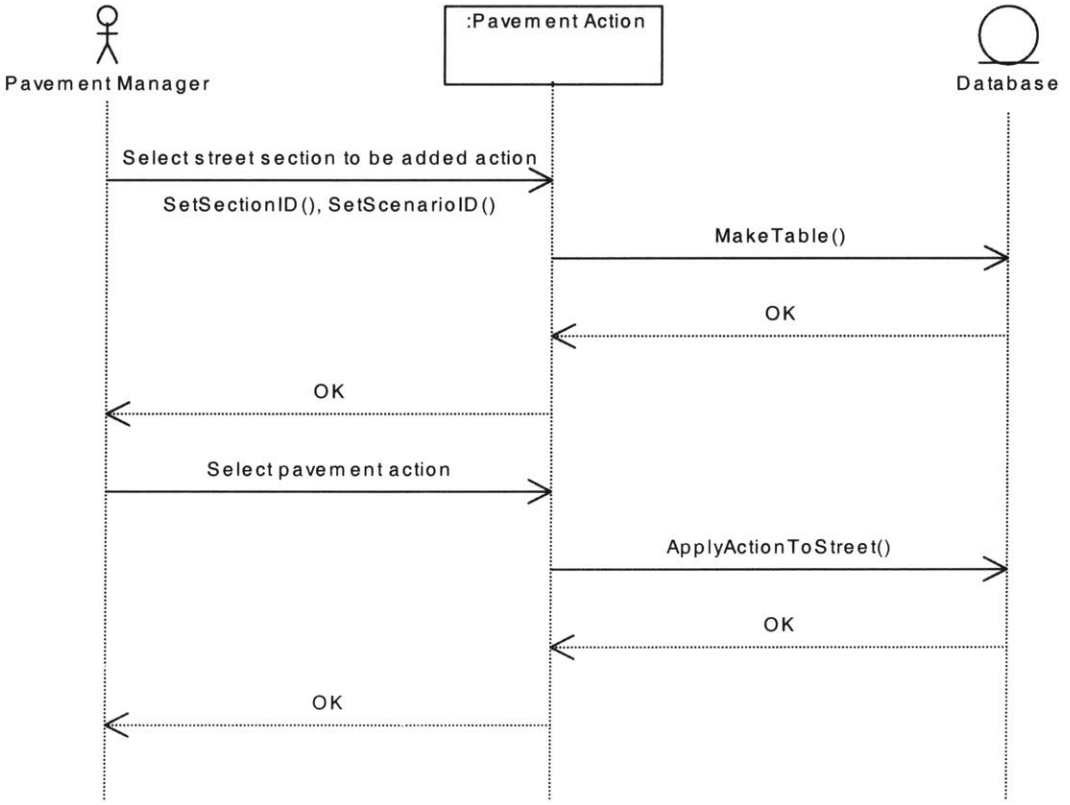


Figure 3-31: Select Pavement Action Sequence diagrams

Select Curb/Sidewalk Action

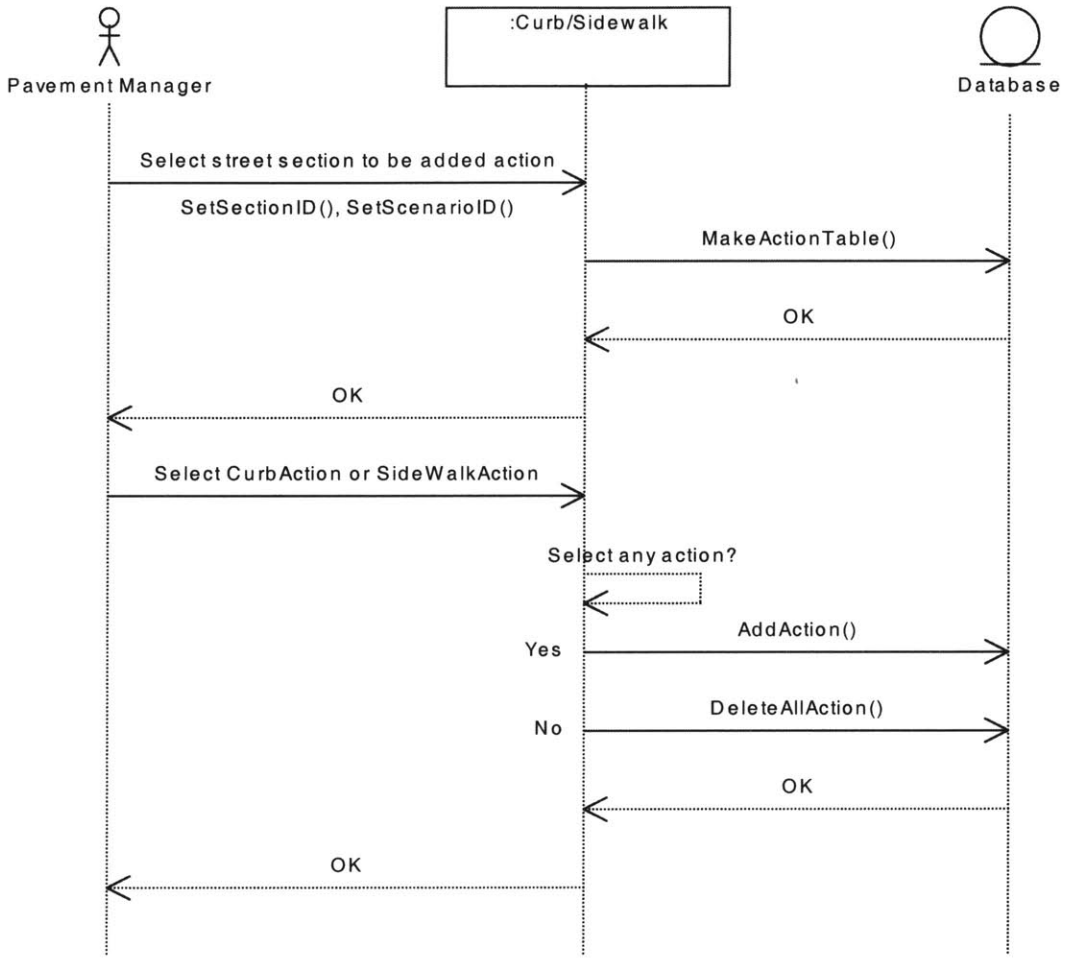


Figure 3-32: Select Curb/Sidewalk Action Sequence diagrams

CHAPTER 4

CONCLUSION

In this chapter, I summarize the essential components needed to expedite the speed of the software development process as well as to achieve a successful project outcome. Future enhancements of this system are indicated as suggestions to improve its quality.

4.1 Essential Components of Development Speed

In order to create the software rapidly and effectively, four important elements must be taken into account and not overlooked. These elements consist of People, Methodology, Tools, and Product.

4.1.1 People

The component that has the most influence to fast development of the project is people (McConnell, 1996). A company or organization that is serious about improving development efficiency should look first at the people issues of motivation, teamwork, and team selection. It can be seen that from the previous chapter that, during the PMIS development process, people are the most critical area that should be considered, as most problems that arose during the process were related to people. Selecting the members with the right skills and talents to take part in the project can prevent these mistakes.

4.1.2 Methodology

The methodology that applies to software development includes both management and technology methodologies. These methodologies specify the process of development and state tasks that must be accomplished, including how they are done and how they relate to

each other. (McConnell, 1996) With the correct process (a.k.a. life cycle models) selection and good management, the project can be performed smoothly and effectively. PMIS development problems resulted from a lack of development fundamentals, especially management fundamentals. Although there was a proper selection of the life cycle model applied to the PMIS project, the project could have progressed more rapidly and successfully if development fundamentals had been considered. More management plans should have been created, and a project management team should have been formed in order to achieve this purpose.

4.1.3 Tools

Technology must have certain characteristics in order to be usable and effective for a successful development process. Choosing the most effective technology to implement the project is a key aspect to initiate the rapid development process. The technology selected to develop the PMIS system depended on the availability of the tools and their affordability to both the teams and the customer. MySQL was chosen to provide the database management system. Since it is freeware and does not offer the fully developed functions of commercial databases, it was sometimes hard to implement the desired task. However, this is considered to be a minor issue, because the other technologies provided good service to develop the project.

4.1.4 Product

Product is the most tangible component from the rest three components. An emphasis on the product size and product features presents a large influence for schedule reduction. (McConnell, 1996) Large projects will take long time to develop, as well as software projects with ambiguous goals and bad-designed scope. Therefore, at the start of the project, it is very important to both the developers and customers to have a unanimous agreement on the characteristics and concepts of the project. In the case study, the customer of the PMIS project knew his requirements, and the objectives of the product were well established at the beginning of the project. However, the use of Web, Palm and

GPS technology were novel elements introduced by the project team that required substantial thought and design activity to ensure a successful product.

4.2 Future Enhancements

Although the PMIS project has ended, there are several concerns that should be addressed to make the system more productive. These concerns are software reusability and maintenance.

Reusability is important to the software development industry nowadays, since software can be used for another project later by notto avoid having to build it from scratch. The reusability of the PMIS system can be improved by providing a well-designed structure of the program including significant comments in the code, and detailed documentation. By performing these tasks, whenever the system need to be maintained or a new, similar system needs to be created, any developer, even one who has not worked on the project before, could understand that code easily. This will shorten any future development schedule.

Since the PMIS system is an academic project, once it ended and was delivered to the customer, there will be no more activity. Therefore, the maintenance phase is omitted from the development process. As stated in chapter two, there may be some changes or flaws that arise to require system changes after delivery. To maintain the system, a team of developers (supported by DPW) should be established to handle requested modifications and errors. Then the development process will be performed again to bring more robustness and reliability to the system.

REFERENCES

Alhir, Sinan S. UML in A Nutshell. California: O'Really, 1998.

Biggerstaff, Ted J. and Charles Richter. Software Reusability Volume I: Concepts and Models. New York: Addison-Wesley, 1989.

Fowlor, Martin, and Kendall Scott. UML Distilled Second Edition. New Jersey: Addison-Wesley, 2000.

Jalote, P. An integrated approach to software engineering. New York: Springer, 1997.

Martin, James. Rapid Application Development. New York: Macmillan, 1991.

McConnell, Steve. Rapid Development: Taming wild software schedules. Washington: Microsoft, 1996.

Pressman, Roger S. A Manager's Guide to Software Engineering. New York: McGraw-Hill, 1993.

---. Software Engineering: A Practitioner's Approach. New York: McGraw-Hill, 2000.

Vliet, Hans V. Software Engineering Principles and Practice. New York: John Wiley & Sons, 2000.

Wang, Yingxu, and Graham King. Software Engineering Process: Principles and Application. Florida: CRC Press LLC, 2000.

Cheung, William. Evaluation of Infrastructure Monitoring System Using PDA and GPS Technologies, 2001.

Choi, Yatlun. Evolution of Platform and User Interface in Infrastructure Management System with Case Study of Arlington Pavement Management System, 2001.

Yim, Wai Kei. Data Modeling in a Pavement Management System, 2001.