Multi-database support in the recursive multi-threaded software process management tool

Yi-Chiun Kuo

Follow this and additional works at: https://scholarworks.lib.csusb.edu/etd-project

Part of the Software Engineering Commons

Recommended Citation
https://scholarworks.lib.csusb.edu/etd-project/2266

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.
MULTI-DATABASE SUPPORT IN THE RECURSIVE MULTI-THREADED
SOFTWARE PROCESS MANAGEMENT TOOL

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Yi-Chiun Kuo
March 2002
MULTI-DATABASE SUPPORT IN THE RECURSIVE MULTI-THREADED SOFTWARE PROCESS MANAGEMENT TOOL

A Project
Presented to the
Faculty of
California State University,
San Bernardino

by
Yi-Chiun Kuo
March 2002

Approved by:

Arturo Concepcion, Chair, Computer Science

Joséphine Mendiza

Ernesto Gomez

05 Mar 2002
ABSTRACT

The Recursive Multi-Threaded (RMT) Software Process Management Tool gives software developers the following capabilities: break a large project into a sequence of prototypes (or threads), track these threads individually, and estimate the progress and completion date of the project from these individual threads. The RMT Tool also facilitates team communications by offering all team members and project managers/leaders instant access to information of software artifacts of the project on the Internet. The goal of the project is to provide the RMT Tool with an ability to support multi-database for collaborative software development. In collaborative software development environment, team members from different geographical locations may be developing a single very large project and information about the status and progress of the different components may be needed. The RMT Tool user will have the illusion of a single database access even though there could be more than one database involved in the process. As a demonstration, actual data are used from several previous Algorithma projects of Software Engineering class taught in the Department of Computer Science in Winter 2000 and Winter 2001.
TABLE OF CONTENTS

ABSTRACT................................................................. iii
LIST OF TABLES........................................................... vii
LIST OF FIGURES......................................................... viii

CHAPTER ONE: SOFTWARE REQUIREMENT SPECIFICATION

Introduction............................................................... 1
  History................................................................. 1
  Purpose................................................................. 2
  Scope.................................................................. 2
  Definitions, Acronyms and Abbreviations.............. 3
  Document Overview.................................................. 5
Overall Description.................................................... 5
  Recursive Multi-Threaded Overview.................. 5
  Product Perspective................................................. 9
  Product Functions.................................................... 11
  User Characteristics................................................. 12
  Constraints............................................................ 12
  Assumptions and Dependencies..................... 13
Specific Requirements............................................ 13
  External Interfaces................................................. 13
  Functions............................................................. 14
  Performance Requirements.......................... 15
  Logical Database Requirements.................. 15
  Design Constraints............................................. 16
CHAPTER FIVE: MAINTENANCE MANUAL

Source Code........................................... 64
Re-Compile........................................... 65
Installation Process............................... 65

CHAPTER SIX: CONCLUSION AND FUTURE DIRECTION........... 67

APPENDIX A: PSEUDO CODE.............................. 70
APPENDIX B: CHECK LIST................................. 93
REFERENCES........................................... 98
LIST OF TABLES

Table 2.1. Personnel Table ........................................ 19
Table 2.2. Position Table .......................................... 19
Table 2.3. Project Table ........................................... 20
Table 2.4. Thread Table ............................................ 21
Table 2.5. Assignment Table ...................................... 22
Table 2.6. Project_Checklist Table ............................... 23
Table 2.7. Thread_Checklist Table ................................. 24
Table 2.8. Programmer_Checklist Table ......................... 26
Table 2.9. Iteration Table ......................................... 27
Table 2.10. Iteration Phase Table ............................... 28
Table 3.1. The Results of the Unit Test ....................... 46
Table 3.2. The Results of the Integration Test .............. 48
LIST OF FIGURES

Figure 1.1. Multi-Database Support ......................... 3
Figure 1.2. Interface for Input ................................ 7
Figure 1.3. Deployment Diagram of Multi-Database ........ 10
Figure 1.4. Use Case Diagram .................................. 12
Figure 1.5. Interface for Output ............................... 14
Figure 2.1. Entity-Relationship Diagram ..................... 17
Figure 2.2. Class Diagram ....................................... 30
Figure 2.3. ConnPool Class Diagram ......................... 31
Figure 2.4. SQLBridge Class Diagram .......................... 32
Figure 2.5. Project Class Diagram ............................. 33
Figure 2.6. Assignment Class Diagram ......................... 34
Figure 2.7. Personnel Class Diagram ........................... 35
Figure 2.8. Thread Class Diagram ............................... 36
Figure 2.9. Programmer_Checklist Class Diagram ............ 37
Figure 2.10. Position Class Diagram ........................... 39
Figure 2.11. Iteration Class Diagram ......................... 40
Figure 2.12. Iteration_Phase Class Diagram ................... 40
Figure 2.13. Thread_Checklist Class Diagram ................ 41
Figure 2.14. Project_Checklist Class Diagram ................ 43
Figure 4.1. The Main Structure of Demonstrate Window ........ 49
Figure 4.2. Information Window for Whole Project ........... 50
Figure 4.3. Single Project View ................................. 51
Figure 4.4. Thread Information Page ......................... 52
Figure 4.5. Update Thread Information Page............... 53
Figure 4.6. Thread Members Page......................... 54
Figure 4.7. Personnel Information Page.................... 55
Figure 4.8. Checklist Page................................. 56
Figure 4.9. Overall Progress Page......................... 57
Figure 4.10. Specification Page............................ 58
Figure 4.11. Planning Page................................. 59
Figure 4.12. Analysis Page................................. 60
Figure 4.13. Design Page................................. 61
Figure 4.14. Implementation Page......................... 62
Figure 4.15. Testing Page................................. 63
CHAPTER ONE
SOFTWARE REQUIREMENT SPECIFICATION

Introduction

History

Recursive Multi-Threaded (RMT) Software Life Cycle is an object-oriented software life cycle developed as a master’s thesis by Scott Simon, and is implemented by Chung-Ping Lin as his master’s project. The RMT Tool is a computer aided software engineering tool for monitoring and predicting software development process. This project improves Lin’s project through the following functions:

- Multi-database support: Lin’s project only supports one database while this project provides the ability for accessing several databases at once - there are three databases used in the demonstration.

- Relational database support and design: mini-SQL database was used in Lin’s project while this project used the SQL Server 7.0 and Oracle 8.

- Additional database functions: deletion, modification, and tables join operation, which were not supported in Lin’s project.
Purpose

This Software Requirements Specification (SRS) defines the functionality of the multi-database in the Recursive Multi-Threaded Software Process Management Tool (Simon and Lin). While the product primarily targets a specialized technical audience, this specification is presented in a form suitable for technical and semi-technical personnel.

This SRS provides the following functionalities:

- Provides a basis for requirement and resource estimation.
- Provides a baseline for validation and verification.

The goal of this project is to provide a multi-database support for RMT Software Process Management Tool.

Scope

The RMT Software Process Management Tool is an application that incorporates the RMT philosophy into an easy to use Web-based interface. The primary purpose of the tool is to allow software developers to break a large project into smaller pieces (or threads), to track these threads individually, and to estimate the progress and completion date from these individual threads. The RMT tool also facilitates team communication by offering all team
members instant access to information about every piece of their project.

This project, multi-database in RMT, will build a multi-database system to support RMT functionalities, see Figure 1.1. A commercial version of RMT interface will be written by another project (Zamora).

![Figure 1.1. Multi-Database Support](image)

**Definitions, Acronyms and Abbreviations**

This section defines terms, acronyms, and abbreviations used in this SRS.

**ER diagram.** Diagrams that use Entity-Relationship model to design or describe database.

**Java Servlet.** Java servlets are a key component of server-side Java development. A servlet is a small, pluggable
extension to a server that enhances the server's functionality.

**JDBC.** Java Database Connectivity is a Java Application Interface that provides Java programmers with a uniform interface for accessing and manipulating a wide range of relational databases.

**Multi-Database System.** A multi-database system is an environment in which data stored in two or more database instances are accessible as though these data were in a single instance.

**PSP.** Personal Software Progress (PSP) is a self-improvement process designed to help the developer controls, manage, and improve the development process. It is a structured framework of forms, guidelines, and procedures for developing software.

**RMT.** Recursive Multi-Threaded (RMT) is a software lifecycle model, and was the result of the master's project of Scott Simon at California State University San Bernardino. RMT is a thread-based approach to organize software development and to monitor developing progress. RMT is a milestone-based, iterative lifecycle that supports incremental and parallel development.
Document Overview

Section One includes the RMT GUI specifications. It also includes multi-database product perspective, product functions, user characteristics, constraints and assumptions, and dependencies. It provides the background and requirements for the whole multi-database project.

In Section Two, the specific requirements of the multi-database external interfaces, functions, performance requirements, logical database requirements, design constraints and software system attributes will be discussed.

Overall Description

Recursive Multi-Threaded Overview

The RMT user is likely to be a technical (developer) or semi-technical (QA staff, project manager) individuals familiar with computing terms. As such, technical aspects of the RMT system are exposed to the user to offer maximum functionality.

The RMT system is a Web-based application that allows developers, development managers, product managers and others involved in the development process to assess the current status of a project. The RMT tool will run on any platform containing browsers that support HTML V.3.2 and
JavaScript. As a Web-based application, interface elements such as edit fields, combo boxes and other common windows GUI controls will retain the look and feel of their parent platform.

RMT uses the concept of frames to organize data representation. The basic RMT window consists of a header section appearing on top of the display window, a navigation window appearing on the left and a view window appearing on the right. The change in the contents of the view window is based on the navigational tree node selected on the left part of the window as shown in Figure 1.2.

The RMT interface consists of three main components: personnel, projects and project groups. Each section is represented as a navigation tree on the left side of the interface. There is also a login interface and a help system. All interfaces are described in detail in the RMT Tool Software Requirements Section below.

RMT relies on HTML V.3.2 compliant browsers to provide navigation, display, text and image rendering, and basic print capabilities. The client side browser will manage all GUI requirements for RMT. RMT will run on any operating system platform containing browsers that support HTML V.3.2 and JavaScript.
The server side portion of the RMT tool requires a platform that offers a Web server and Java Servlet runner. The current iteration of RMT is designed to run on the Linux platform using JRun Java Servlet engine.

RMT provides the following basic functionalities:

- Allows a development manager to create project 'threads,' or pieces of project functionality to be assigned to a project sub team.
- Allow individual developers to assess and enter thread progress, along with thread documentation.
• Allows the project manager to view total project status based on values entered by individual developers.

• Provides a framework for the PSP Scriber subsystem (Tsao).

• Allows navigation between various RMT forms and interfaces.

• Allows printing individual forms using the browser print function.

RMT is primarily designed for software engineers and managers. These users are likely to be technically competent and familiar with browser technology. No special consideration is given to documenting the use of browser technology in RMT. In addition, it is assumed that the RMT user is familiar with windowing interfaces, edit boxes, combo boxes, and other controls common in windowing interfaces. No special consideration is given to documenting windowing interfaces use or navigation.

The user may not be familiar with RMT. Because of this, RMT needs to clearly state how each form is used, supply help and examples, guide the user into the proper form based on need.
RMT interfaces are based on HTML V.3.2 and client side JavaScript supported by the client browser. Because of this, RMT does not require any functionality that relies on client-side executable code outside of the browser environment.

RMT relies on tables, and will only run on a browser that is HTML V.3.2 compliant.

The RMT server relies on Java, Java Servlet and Oracle. While many platforms offer these functionalities, RMT is currently developed for the Linux platform. As such, certain additional work may be necessary to move the RMT server to other platforms.

While increasingly unlikely, some RMT users may connect to the server via modem, consideration should be given to page sizes, image complexity, and other factors that affect download speed.

Product Perspective

User Interface. The user for this project is the RMT project. The user interface for the multi-database project are JDBC classes. Since the JDBC classes are written in object-oriented approach, my teammate, who is writing the user interface part of the RMT project, does not have to know how the database part is implemented to access the database.
Hardware Interfaces. The server operating system manages hardware interface issues. There is no known hardware related hardware interface issues.

Software Interfaces. This project relies on Oracle 8.1.6 database and JDBC classes. Therefore the server side portion of this project requires a platform that offers a Web server and Java Servlet runner. The current iteration of RMT is designed to run on the Linux platform using JRun Java-Servlet engine.

Figure 1.3 shows the system interfaces of the multi-databases.
Communication Interfaces. RMT relies on the client operating system, and the server Java Servlet engine and operating system to manage communication issues.

Memory and Hardware Constraints. There is no known RMT client-side memory constraint issues.

Additionally, the server requires the following components for optimal content generation in a multi-user environment.

- 128 MB or greater memory size.

- Fast architecture machine, something along the lines of a PII 500 or greater.

Adaptation Requirements. This project has no known site adaptation at this time.

Product Functions

There are two main functionalities provided by this project. One is to allow RMT users to create, delete, update and retrieve projects, threads, personnel and iteration information from the database. This functionality is shown in Figure 1.4. The other one is to support multiple sites access showing only a single RMT hierarchical tree. In other words, an RMT project can store its information in different databases at different locations. Once an RMT user wants to use data that is not
stored in the local database, RMT can still get the desired information from the appropriate site through multi-database support.

**User Characteristics**

This project is primarily designed for the RMT user.

**Constraints**

The server relies on Java, Java Servlet and Oracle. While many platforms offer these functionalities, this project is currently developed for the Linux platform. As
such, certain additional work may be necessary to port this project server to other platforms or other databases.

Assumptions and Dependencies

This proposal makes the following assumptions:

- The same relation (table) types exist for all the databases.
- Additionally, it is assumed that the server platform is a Linux system with the following components installed and working properly:
  - Apache 1.3.6-7 or other web server system
  - Allaire JRun Servlet Runner 2.3 or greater
  - Oracle-8.1.6
  - Redhat Package manager (rpm) or similar installation system
  - JDK 1.2.2

Specific Requirements

External Interfaces

Figure 1.5 shows the user interface of the RMT project. RMT uses this interface to get information from the user or displays the information onto the interface. The RMT project has to use JDBC classes to store or retrieve from the database.
Functions

A big company normally has several geographically located branches and each branch usually has its own database, which are accessed locally unless there is a user that requests some remote data from other branches via the Internet.

This project will use the Oracle database management system to support management of software projects. The multi-database allows insertion, retrieval, deletion and update of information regarding software projects stored in
Moreover, the RMT project also requires an interface to interact with the Oracle database. Since the user interface in RMT project is using Java Servlet, it is better to use the same language for the interface between the user interface and the databases. Therefore, JDBC was chosen for this interface.

In JDBC classes, there is a class to handle the driver of the database so that databases at different locations can support the RMT project management software. The other JDBC classes provide the deletion, selection, insertion and update functions. The RMT project management software can insert and retrieve data from the database via these classes.

**Performance Requirements**

The number of databases to be supported in the demonstration is three but the number of databases that can be supported is a large number. The number of simultaneous users and the data to be supported depends on the traffic on the Internet.

**Logical Database Requirements**

There are ten tables created to store all the information on the RMT process management tool - they are Personnel, Position, Project, Thread, Assignment,
Programmer_checklist, Thread_checklist, Project_checklist, Iteration and Iteration Phase.

Design Constraints

There are no design constraints at this time.

Software System Attributes

Security and Reliability. JDBC supports safe programming practices on a number of levels. Because they are written in Java, JDBC's inherit the strong type safety of the Java language.

Portability and Availability. Because JDBC classes are written in Java and conforms to a well-defined and widely accepted database connection, they are highly portable across operating systems and across server implementations. They can be used on a Windows NT machine running Java Web Server and later deployed effortlessly on a high-end Unix server running Apache. With JDBC, one can truly "write once, serve everywhere."
CHAPTER TWO
DESIGN

Database Design

There are three databases; two Oracle databases and one Microsoft SQL Server database, used in the multi-database project. One Oracle database is installed on the RMT server (rmt.ias.csusb.edu), named rmt and the other Oracle database is located in give2you.net server, named premio. The SQL Server is installed on give2you.net server.

Table Design

The information on the RMT tool are stored in ten tables. The E-R diagram (Figure 2.1) represents the

Figure 2.1. Entity-Relationship Diagram
relationships among these ten tables. A Project may run on one or more Threads. A Thread could be implemented by one or more Iterations. An Iteration may have at most seven Iteration Phases. A Personnel can be assigned to one or more Threads and a Thread may have one or more programmer assigned to it. There are several Position levels when a Personnel is assigned to several Threads. In the meanwhile, there is a corresponding checklist generated for the Personnel depending on its Position ID.

This multi-database project used foreign key constraints on some tables to make data consistent. In Figure 2.1 Entity-Relationship Diagram, Personnel, Project, Thread, Checklist, Iteration, and Iteration_Phase are entities. Therefore, each of them needs a table. The detailed information about each data entity and their relationships are listed in Table 2.1 through Table 2.8.

Personnel table contains information about the different persons involved in a project, such as the project leader, the team leader and programmers. The detailed information for the columns of Personnel table is shown in Table 2.1.
Table 2.1. Personnel Table

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee ID</td>
<td>varchar</td>
<td>15</td>
<td>User login ID (Primary key)</td>
</tr>
<tr>
<td>First Name</td>
<td>varchar</td>
<td>15</td>
<td>User first name</td>
</tr>
<tr>
<td>Last Name</td>
<td>varchar</td>
<td>15</td>
<td>User last name</td>
</tr>
<tr>
<td>Phone Number1</td>
<td>varchar</td>
<td>15</td>
<td>User contact phone number</td>
</tr>
<tr>
<td>Phone Number2</td>
<td>varchar</td>
<td>15</td>
<td>User second contact phone number or mobile phone number</td>
</tr>
<tr>
<td>Email</td>
<td>varchar</td>
<td>20</td>
<td>User email address</td>
</tr>
<tr>
<td>Password</td>
<td>varchar</td>
<td>10</td>
<td>User password which is chosen by user</td>
</tr>
</tbody>
</table>

A Personnel assigned to a Thread is given a Position ID. Position table (Table 2.2) is needed to store Position ID and its description. The primary key of Position table is Position ID.

Table 2.2. Position Table

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position ID</td>
<td>varchar</td>
<td>2</td>
<td>Position ID (Primary key)</td>
</tr>
<tr>
<td>Description</td>
<td>varchar</td>
<td>20</td>
<td>Description for Position</td>
</tr>
</tbody>
</table>
### Table 2.3. Project Table

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project ID</td>
<td>varchar</td>
<td>10</td>
<td>Project ID (Primary key)</td>
</tr>
<tr>
<td>Project Name</td>
<td>varchar</td>
<td>20</td>
<td>The name of project</td>
</tr>
<tr>
<td>Starting Date</td>
<td>varchar</td>
<td>10</td>
<td>The date when project created</td>
</tr>
<tr>
<td>Estimate Duration</td>
<td>varchar</td>
<td>4</td>
<td>The estimate duration of the project</td>
</tr>
<tr>
<td>Progress</td>
<td>float</td>
<td>4</td>
<td>Overall progress of the project</td>
</tr>
</tbody>
</table>

Project table (Table 2.3) describes status information about the project. Project ID, project name, starting date, and estimate duration are provided by the project leader. The progress of the project is computed automatically when any change is made on Project_checklist.

Thread table describes status information about the thread. Thread ID is generated automatically according the last thread ID. Project ID is given according which project this thread is belong to. Progress is computed when any change is made on thread_checklist by thread leader. Total Iteration is estimated and input by team leader. Actual Total Iteration is calculated when adding or deleting an iteration. Mother Iteration is assigned according its
mother thread's ID. Thread ID is a primary key of Thread table. Table 2.4 contains the detailed information for Thread table.

Table 2.4. Thread Table

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread ID</td>
<td>varchar</td>
<td>10</td>
<td>Thread ID (Primary key)</td>
</tr>
<tr>
<td>Project ID</td>
<td>varchar</td>
<td>10</td>
<td>The ID of the project running on the thread</td>
</tr>
<tr>
<td>Progress</td>
<td>float</td>
<td>4</td>
<td>The progress of the thread</td>
</tr>
<tr>
<td>Total Iteration</td>
<td>int</td>
<td>1</td>
<td>The estimate total iteration number of the thread</td>
</tr>
<tr>
<td>Actual Total Iteration</td>
<td>int</td>
<td>1</td>
<td>The actual total iteration number of the thread</td>
</tr>
<tr>
<td>Mother Thread</td>
<td>varchar</td>
<td>10</td>
<td>The mother thread ID of this thread</td>
</tr>
</tbody>
</table>

Since a Personnel can be assigned to many Threads and one Thread can have many Personnel, the relationship between Personnel and Thread is many-to-many relationship. Because of this an Assignment table is needed. As shown in Table 2.5, the assignment table indicated which personnel is assigned to a thread.
Table 2.5. Assignment Table

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project ID</td>
<td>varchar</td>
<td>10</td>
<td>The project ID of the assignment (Primary key component)</td>
</tr>
<tr>
<td>Employee ID</td>
<td>varchar</td>
<td>10</td>
<td>The employee ID of the assignment (Primary key component)</td>
</tr>
<tr>
<td>Thread ID</td>
<td>varchar</td>
<td>10</td>
<td>The thread ID of the assignment (Primary key component)</td>
</tr>
<tr>
<td>Position ID</td>
<td>varchar</td>
<td>2</td>
<td>The position ID of the assignment</td>
</tr>
</tbody>
</table>

When a Personnel is assigned to a Thread, the Personnel will need a Checklist to record the work progress. A Personnel can be a project leader, a team leader, or a programmer; therefore, as shown in Figure 2.1, the actual Checklist created should be one of the following: Project Leader Checklist, Team Leader Checklist, or Programmer Checklist. In other words, a Project Checklist table, a Thread Checklist table, and a Programmer Checklist table are needed.

Project_Checklist table (Table 2.6) stored all project progress checkpoints. There are seventeen checkpoints in this table. The default value of these checkpoints are
zero. If one checkpoint is done, it should be checked and the checked value is one.

Table 2.6. Project_Checklist Table

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project ID</td>
<td>varchar</td>
<td>10</td>
<td>The project ID used by the project leader (Primary key component)</td>
</tr>
<tr>
<td>Employee ID</td>
<td>varchar</td>
<td>20</td>
<td>The employee ID of the project leader (Primary key component)</td>
</tr>
<tr>
<td>C1</td>
<td>Number</td>
<td>4</td>
<td>Problem/System Definition</td>
</tr>
<tr>
<td>C2</td>
<td>Number</td>
<td>4</td>
<td>System Environment</td>
</tr>
<tr>
<td>C3</td>
<td>Number</td>
<td>4</td>
<td>Interface Definition</td>
</tr>
<tr>
<td>C4</td>
<td>Number</td>
<td>4</td>
<td>Human Resource/Engineering Requirement</td>
</tr>
<tr>
<td>C5</td>
<td>Number</td>
<td>4</td>
<td>Producing SQA</td>
</tr>
<tr>
<td>C6</td>
<td>Number</td>
<td>4</td>
<td>Producing SPMP</td>
</tr>
<tr>
<td>C7</td>
<td>Number</td>
<td>4</td>
<td>Use Case Diagram</td>
</tr>
<tr>
<td>C8</td>
<td>Number</td>
<td>4</td>
<td>Sequence Diagram</td>
</tr>
<tr>
<td>C9</td>
<td>Number</td>
<td>4</td>
<td>Class Diagram</td>
</tr>
<tr>
<td>C10</td>
<td>Number</td>
<td>4</td>
<td>Complent Diagram</td>
</tr>
<tr>
<td>C11</td>
<td>Number</td>
<td>4</td>
<td>Interaction Diagram</td>
</tr>
<tr>
<td>C12</td>
<td>Number</td>
<td>4</td>
<td>State Diagram</td>
</tr>
<tr>
<td>C13</td>
<td>Number</td>
<td>4</td>
<td>Architectural Design</td>
</tr>
<tr>
<td>C14</td>
<td>Number</td>
<td>4</td>
<td>Distributing Object Modules to Team</td>
</tr>
</tbody>
</table>
Table 2.6. Project_Checklist Table (cont.)

<table>
<thead>
<tr>
<th></th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C15</td>
<td>4</td>
<td>Collecting Object Modules From Team (Waiting for)</td>
</tr>
<tr>
<td>C16</td>
<td>4</td>
<td>Team Level Unit Test</td>
</tr>
<tr>
<td>C17</td>
<td>4</td>
<td>System Wide Integration Test</td>
</tr>
</tbody>
</table>

Thread_Checklist table includes sixteen check points of thread progress. See Table 2.7 for the detail of the checklist items listed in C1 to C16. The usage of the checklist table is the same as Project Checklist table.

Table 2.7. Thread_Checklist Table

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread ID</td>
<td>varchar</td>
<td>10</td>
<td>The thread ID used by the team leader</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Primary key component)</td>
</tr>
<tr>
<td>Employee ID</td>
<td>varchar</td>
<td>20</td>
<td>The employee ID of the team leader</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Primary key component)</td>
</tr>
<tr>
<td>Level ID</td>
<td>varchar</td>
<td>10</td>
<td>The position ID of the team leader</td>
</tr>
<tr>
<td>Iteration ID</td>
<td>varchar</td>
<td>10</td>
<td>The iteration ID of the team leader</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Primary key component)</td>
</tr>
<tr>
<td>C1</td>
<td>Number</td>
<td>4</td>
<td>Study and Understand SRS</td>
</tr>
<tr>
<td>C2</td>
<td>Number</td>
<td>4</td>
<td>Specify Team Level Requirement</td>
</tr>
<tr>
<td>C3</td>
<td>Number</td>
<td>4</td>
<td>Study and Understand SQA</td>
</tr>
</tbody>
</table>
Table 2.7. Thread_Checklist Table (cont.)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C4</td>
<td>Number</td>
<td>4</td>
<td>Team Level Development Schedule Based on SPMP</td>
</tr>
<tr>
<td>C5</td>
<td>Number</td>
<td>4</td>
<td>Organize Programmers Based on SPMP</td>
</tr>
<tr>
<td>C6</td>
<td>Number</td>
<td>4</td>
<td>Decomposing Object</td>
</tr>
<tr>
<td>C7</td>
<td>Number</td>
<td>4</td>
<td>Class Diagram</td>
</tr>
<tr>
<td>C8</td>
<td>Number</td>
<td>4</td>
<td>Component Diagram</td>
</tr>
<tr>
<td>C9</td>
<td>Number</td>
<td>4</td>
<td>Interaction Diagram</td>
</tr>
<tr>
<td>C10</td>
<td>Number</td>
<td>4</td>
<td>State Diagram</td>
</tr>
<tr>
<td>C11</td>
<td>Number</td>
<td>4</td>
<td>Number of Objects Completed</td>
</tr>
<tr>
<td>C12</td>
<td>Number</td>
<td>4</td>
<td>Object Relation Definition</td>
</tr>
<tr>
<td>C13</td>
<td>Number</td>
<td>4</td>
<td>Distributing Object Modules to Programmer</td>
</tr>
<tr>
<td>C14</td>
<td>Number</td>
<td>4</td>
<td>Collecting Object Modules From Programmer</td>
</tr>
<tr>
<td>C15</td>
<td>Number</td>
<td>4</td>
<td>Team Level Unit Done</td>
</tr>
<tr>
<td>C16</td>
<td>Number</td>
<td>4</td>
<td>Team Level Integration Test</td>
</tr>
</tbody>
</table>

Programmer_Checklist table contains sixteen columns to store the check points of programmer work progress. Table 2.8 lists the check point description from C1 to C16. The check point value is the same as Project Checklist table.
### Table 2.8. Programmer Checklist Table

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread ID</td>
<td>varchar</td>
<td>10</td>
<td>The thread ID used by the programmer (Primary key component)</td>
</tr>
<tr>
<td>Employee ID</td>
<td>varchar</td>
<td>20</td>
<td>The employee ID of the programmer (Primary key component)</td>
</tr>
<tr>
<td>Iteration ID</td>
<td>varchar</td>
<td>10</td>
<td>The iteration ID of the programmer (Primary key component)</td>
</tr>
<tr>
<td>C1</td>
<td>Number</td>
<td>4</td>
<td>Study and Understand SRS and SQL</td>
</tr>
<tr>
<td>C2</td>
<td>Number</td>
<td>4</td>
<td>Specify Requirement for Object</td>
</tr>
<tr>
<td>C3</td>
<td>Number</td>
<td>4</td>
<td>Programmer’s SQL Level</td>
</tr>
<tr>
<td>C4</td>
<td>Number</td>
<td>4</td>
<td>Programmer Development Schedule Level Based on SPMP</td>
</tr>
<tr>
<td>C5</td>
<td>Number</td>
<td>4</td>
<td>Modified Object</td>
</tr>
<tr>
<td>C6</td>
<td>Number</td>
<td>4</td>
<td>Deleted Object</td>
</tr>
<tr>
<td>C7</td>
<td>Number</td>
<td>4</td>
<td>Added Object</td>
</tr>
<tr>
<td>C8</td>
<td>Number</td>
<td>4</td>
<td>Reused Object</td>
</tr>
<tr>
<td>C9</td>
<td>Number</td>
<td>4</td>
<td>Object Pseudo Code</td>
</tr>
<tr>
<td>C10</td>
<td>Number</td>
<td>4</td>
<td>Object Design - Number of Completed</td>
</tr>
<tr>
<td>C11</td>
<td>Number</td>
<td>4</td>
<td>Modified Object</td>
</tr>
<tr>
<td>C12</td>
<td>Number</td>
<td>4</td>
<td>Deleted Object</td>
</tr>
<tr>
<td>C13</td>
<td>Number</td>
<td>4</td>
<td>Added Object</td>
</tr>
</tbody>
</table>
Table 2.8 Programmer_Checklist Table (cont.)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C14</td>
<td>Number</td>
<td>4</td>
<td>Reused Object</td>
</tr>
<tr>
<td>C15</td>
<td>Number</td>
<td>4</td>
<td>Object Unit Test</td>
</tr>
<tr>
<td>C16</td>
<td>Number</td>
<td>4</td>
<td>Programmer Integration Level</td>
</tr>
</tbody>
</table>

A Thread may be implemented by one or more Iteration. Therefore, Iteration table is needed to record this information. Iteration table describes status information about the iteration. Thread ID and Iteration ID are the primary keys of Iteration table, see Table 2.9.

Table 2.9. Iteration Table

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread ID</td>
<td>varchar</td>
<td>10</td>
<td>The thread ID of the iteration (Primary key component)</td>
</tr>
<tr>
<td>Iteration ID</td>
<td>varchar</td>
<td>10</td>
<td>The iteration ID of the iteration (Primary key component)</td>
</tr>
<tr>
<td>Weight</td>
<td>float</td>
<td>4</td>
<td>The weight of the iteration</td>
</tr>
</tbody>
</table>

An Iteration can have up to seven Iteration Phases. As shown in Figure 2.1, Iteration Phase is an entity so an Iteration Phase table is needed. Iteration Phase table store the progress and document URL of each iteration phase. The data type of these four columns is varchar.
Iteration table uses Iteration ID and Phase as primary keys. The Iteration table is shown in Table 2.10.

Table 2.10. Iteration Phase Table

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration ID</td>
<td>varchar</td>
<td>10</td>
<td>Iteration ID of the iteration phase (Primary key component)</td>
</tr>
<tr>
<td>Phase</td>
<td>varchar</td>
<td>10</td>
<td>The phase of the iteration phase (Primary key component)</td>
</tr>
<tr>
<td>Progress</td>
<td>float</td>
<td>4</td>
<td>The progress of the iteration phase</td>
</tr>
<tr>
<td>Document URL</td>
<td>varchar</td>
<td>200</td>
<td>The document of the iteration phase</td>
</tr>
</tbody>
</table>

Database Access

Using the database links created between the two Oracle databases, the RMT interface can access the databases remotely via the local database. Moreover, each table in the Oracle database installed in rmt.ias.csusb.edu server has a snapshot in the Oracle database installed in give2you.net server.

The snapshot is the redundant data for the remote database (rmt.ias.csusb.edu) and is stored in the local database server (give2you.net). By using these snapshots, it will be more efficient than accessing the data directly.
from the remote database (rmt.ias.csusb.edu). In addition, once the remote database is down, these snapshots will still have the redundant data for backup. However, Microsoft SQL Server installed on give2you.net uses JDBC class to access its own databases directly.

Software Architecture

The user interface of RMT is developed using Java programming language. In order to gain the best compatibility, the connection interface between the user interface and the databases is developed using JDBC class, which is a member of the Java programming language family.

There are ten classes (Figure 2.2) used in the multi-database project. Each of the ten tables has its own class to insert or retrieve data from the databases. SQLBridge class use ConnPool to open or close database. Every class extends from SQLBridge class to execute a SQL statement.

SQLbridge is used for executing all the SQL statements. This class has four purposes: connecting to a database, executing a SQL statement, receiving and processing the result, and closing a database connection.

ConnPool is used to make a connection pool, and every request for database must go through it. In other words, each request for database does not establish its own
connection. In this way, we can reduce the number of connections established and make the system more efficient. The maximum number of connections that can be opened simultaneously is three. Once the fourth request comes in, it has to wait until one of these three connections has been returned back to the connection pool.

The rest of the classes are used to access the appropriate table. Each of these classes has insert(), delete(), update(), and select() methods to retrieve or
update data. If some errors occur while executing any of these methods, an error message is returned.

Detailed Design

This section will list all ten classes with their class diagram. See Appendix A for the pseudocode of all the methods in each Class.

ConnPool Class

The purpose of ConnPool class (Figure 2.3) is to make a connection pool.

```
ConnPool
private Vector freeConnections
private Hashtable boundedConnections
private String driveName
private String jdbcURL
private String username
private String password
private int maxConnections

public ConnPool()
public void closeDB()
public synchronized Connection getConnection()
public void openDB()
public synchronized void returnConnection()
public void setConnectionSwitch()
.public void setMaxConnections()
.public void setDriverName()
.public void setJDBCURL()
.public void setUsername()
.public void setPassword()
```

Figure 2.3. ConnPool Class Diagram
SQLBridge Class

The purpose of SQLBridge class (Figure 2.4) are to open a database, execute SQL statement, get execute result, and close a database.

```
<table>
<thead>
<tr>
<th>SQLBridge</th>
</tr>
</thead>
<tbody>
<tr>
<td>private ConnPool connPool</td>
</tr>
<tr>
<td>private Connection conn</td>
</tr>
<tr>
<td>private ResultSet rs</td>
</tr>
<tr>
<td>private ResultSetMetaData rsmd</td>
</tr>
<tr>
<td>private Statement stmt</td>
</tr>
<tr>
<td>private String driverName</td>
</tr>
<tr>
<td>private String jdbcURL</td>
</tr>
<tr>
<td>private String username</td>
</tr>
<tr>
<td>private String password</td>
</tr>
<tr>
<td>private String errMsg</td>
</tr>
</tbody>
</table>

public SQLBridge() |
private void clearResult() |
public void closeDB() |
public int execSQL() |
public ResultSet execRecord() |
public void execUpdate() |
public int getColumnCount() |
public String[] getColumnNames() |
protected object getField() |
public Object getField() |
public String getFieldName() |
public Boolean nextRow() |
public void setErrMsg() |
public String getErrMsg() |
public void openDB() |
public void setConnectionSwitch() |
public void setConnPool() |
public void setDriverName() |
public void setJdbcURL() |
public void setUserName() |
public void setPassword() |
```

Figure 2.4. SQLBridge Class Diagram
Project Class

The purpose of Project class (Figure 2.5) is to provide delete, insert, update and select functions to manipulate the data in the Project table.

<table>
<thead>
<tr>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>private String PROJECT_ID</td>
</tr>
<tr>
<td>private String PROJECT_NAME</td>
</tr>
<tr>
<td>private String STARTING_DATE</td>
</tr>
<tr>
<td>private String ESTIMATE_DURATION</td>
</tr>
<tr>
<td>private String PROGRESS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public void Project()</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void setPROJECT_ID()</td>
</tr>
<tr>
<td>public void setPROJECT_NAME()</td>
</tr>
<tr>
<td>public void setSTARTING_DATE()</td>
</tr>
<tr>
<td>public void setESTIMATE_DURATION()</td>
</tr>
<tr>
<td>public void setPROGRESS()</td>
</tr>
<tr>
<td>public String getPROJECT_ID()</td>
</tr>
<tr>
<td>public String getPROJECT_NAME()</td>
</tr>
<tr>
<td>public String getSTARTING_DATE()</td>
</tr>
<tr>
<td>public String getESTIMATE_DURATION()</td>
</tr>
<tr>
<td>public String getPROGRESS()</td>
</tr>
<tr>
<td>public int inert()</td>
</tr>
<tr>
<td>public int delete()</td>
</tr>
<tr>
<td>public ResultSet selectRecord()</td>
</tr>
<tr>
<td>public void select()</td>
</tr>
<tr>
<td>public int update()</td>
</tr>
</tbody>
</table>

Figure 2.5. Project Class Diagram
Assignment

| private String PROJECT_ID |
| private String EMPLOYEE_ID |
| private String THREAD_ID |
| private String POSITION_ID |

| public void Assignment() |
| public void setPROJECT_ID() |
| public void setEMPLOYEE_ID() |
| public void setTHREAD_ID() |
| public void setPOSITION_ID() |
| public String getPROJECT_ID() |
| public String getEMPLOYEE_ID() |
| public String getTHREAD_ID() |
| public String getPOSITION_ID() |
| public int insert() |
| public int delete() |
| public ResultSet selectRecord() |
| public void select() |
| public int update() |

Figure 2.6. Assignment Class Diagram

Assignment Class

The purpose of Assignment class (Figure 2.6) is to provide delete, insert, update and select functions to manipulate the data in the Assignment table.
**Personnel**

<table>
<thead>
<tr>
<th>private String EMPLOYEE_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>private String FIRST_NAME</td>
</tr>
<tr>
<td>private String LAST_NAME</td>
</tr>
<tr>
<td>private String USER_PASSWORD</td>
</tr>
<tr>
<td>private String EMAIL</td>
</tr>
<tr>
<td>private String PHONE1</td>
</tr>
<tr>
<td>private String PHONE2</td>
</tr>
<tr>
<td>private String PASS_PHRASE</td>
</tr>
</tbody>
</table>

```java
void Personnel()
void setEMPLOYEE_ID()
void setFIRST_NAME()
void setLAST_NAME()
void setUser_PASSWORD()
void setEmail()
void setPHONE1()
void setPHONE2()
void setPASS_PHRASE()
String getEMPLOYEE_ID()
String getFIRST_NAME()
String getLast_NAME()
String getUser_PASSWORD()
String setEmail()
String setPHONE1()
String setPHONE2()
String setPASS_PHRASE()
boolean passCheck()
int insert()
int delete()  
ResultSet selectRecord()
void select()
int update()  
```

Figure 2.7. Personnel Class Diagram

**Personnel Class**

The purpose of Personnel class (Figure 2.7) is to provide delete, insert, update and select functions to manipulate the data in the Personnel table.
Thread

private String PROJECT_ID
private String THREAD_ID
private String PROGRESS
private String EMPLOYEE_ID
private String CURRENT_ITERATION
private String MOTHER_THREAD

public void Thread()
public void setPROJECT_ID()
public void setTHREAD_ID()
public void setPROGRESS()
public void setEMPLOYEE_ID()
public void setCURRENT_ITERATION()
public void setMOTHER_THREAD()
public String getPROJECT_ID()
public String getTHREAD_ID()
public String getPROGRESS()
public String getEMPLOYEE_ID()
public String getCURRENT_ITERATION()
public String getMOTHER_THREAD()
public int insert()
public int delete()
public ResultSet selectRecord()
public void select()
public int update()

Figure 2.8. Thread Class Diagram

Thread Class

The purpose of Thread class (Figure 2.8) is to provide delete, insert, update and select functions to manipulate the data in the Thread table.
<table>
<thead>
<tr>
<th>Programmer Checklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>private String THREAD_ID</td>
</tr>
<tr>
<td>private String EMPLOYEE_ID</td>
</tr>
<tr>
<td>private String ITERATION_ID</td>
</tr>
<tr>
<td>private int C1</td>
</tr>
<tr>
<td>private int C2</td>
</tr>
<tr>
<td>private int C3</td>
</tr>
<tr>
<td>private int C4</td>
</tr>
<tr>
<td>private int C5</td>
</tr>
<tr>
<td>private int C6</td>
</tr>
<tr>
<td>private int C7</td>
</tr>
<tr>
<td>private int C8</td>
</tr>
<tr>
<td>private int C9</td>
</tr>
<tr>
<td>private int C10</td>
</tr>
<tr>
<td>private int C11</td>
</tr>
<tr>
<td>private int C12</td>
</tr>
<tr>
<td>private int C13</td>
</tr>
<tr>
<td>private int C14</td>
</tr>
<tr>
<td>private int C15</td>
</tr>
<tr>
<td>private int C16</td>
</tr>
<tr>
<td>public void Programmer_checklist()</td>
</tr>
<tr>
<td>public void setTHREAD_ID()</td>
</tr>
<tr>
<td>public void setEMPLOYEE_ID()</td>
</tr>
<tr>
<td>public void setITERATION_ID()</td>
</tr>
<tr>
<td>public void setC1()</td>
</tr>
<tr>
<td>public void setC2()</td>
</tr>
<tr>
<td>public void setC3()</td>
</tr>
<tr>
<td>public void setC4()</td>
</tr>
<tr>
<td>public void setC5()</td>
</tr>
<tr>
<td>public void setC6()</td>
</tr>
<tr>
<td>public void setC7()</td>
</tr>
<tr>
<td>public void setC8()</td>
</tr>
<tr>
<td>public void setC9()</td>
</tr>
<tr>
<td>public void setC10()</td>
</tr>
<tr>
<td>public void setC11()</td>
</tr>
<tr>
<td>public void setC12()</td>
</tr>
<tr>
<td>public void setC13()</td>
</tr>
<tr>
<td>public void setC14()</td>
</tr>
<tr>
<td>public void setC15()</td>
</tr>
<tr>
<td>public void setC16()</td>
</tr>
<tr>
<td>public String getTHREAD_ID()</td>
</tr>
<tr>
<td>public String getEMPLOYEE_ID()</td>
</tr>
<tr>
<td>public String getITERATION_ID()</td>
</tr>
<tr>
<td>public int getCl()</td>
</tr>
</tbody>
</table>

Figure 2.9. Programmer_Checklist Class Diagram
The purpose of Programmer_Checklist class (Figure 2.9) is to provide delete, insert, update and select functions to manipulate the data in the Programmer_Checklist table.
### Position Class

The purpose of *Position* class (Figure 2.10) is to provide delete, insert, update and select functions to manipulate the data in the *Position* table.

#### Iteration Class

The purpose of *Iteration* class (Figure 2.11) is to provide delete, insert, update and select functions to manipulate the data in the *Iteration* table.

<table>
<thead>
<tr>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>private String POSITION_ID</td>
</tr>
<tr>
<td>private String DESCRIPTION</td>
</tr>
<tr>
<td>public void Position()</td>
</tr>
<tr>
<td>public void setPosition_ID()</td>
</tr>
<tr>
<td>public void setDescription()</td>
</tr>
<tr>
<td>public String getPosition_ID()</td>
</tr>
<tr>
<td>public String getDescription()</td>
</tr>
<tr>
<td>public int insert()</td>
</tr>
<tr>
<td>public int delete()</td>
</tr>
<tr>
<td>public ResultSet selectRecord()</td>
</tr>
<tr>
<td>public ResultSet selectRecord()</td>
</tr>
<tr>
<td>public void select()</td>
</tr>
<tr>
<td>public int update()</td>
</tr>
</tbody>
</table>
Iteration

private String THREAD_ID
private String ITERATION_ID
private String WEIGHT

public void Iteration()
public void setTHREAD_ID()
public void set ITERATION_ID()
public void setWEIGHT()
public String getTHREAD_ID()
public String getITERATION_ID()
public String getWEIGHT()
public int insert()
public int delete()
public ResultSet selectRecord()
public void select()
public int update()

Figure 2.11. Iteration Class Diagram

Iteration_Phase

private String ITERATION_ID
private String PHASE
private String PROGRESS
private String DOCUMENT_URL

public void Iteration_phase()
public void setITERATION_ID()
public void setPHASE()
public void setPROGRESS()
public void setDOCUMENT()
public String getITERATION_ID()
public String getPHASE()
public String getPROGRESS()
public String getDOCUMENT()
public int insert()
public int delete()
public ResultSet selectRecord()
public void select()
public int update()

Figure 2.12. Iteration_Phase Class Diagram
**Iteration Phase Class**

The purpose of Iteration Phase class (Figure 2.12) is to provide delete, insert, update and select functions to manipulate the data in the Iteration Phase table.

**Thread Checklist Class**

The purpose of Team_Checklist class (Figure 2.13) is to provide delete, insert, update and select functions to manipulate the data in the Thread_Checklist table.

<table>
<thead>
<tr>
<th>Thread Checklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>private String THREAD_ID</td>
</tr>
<tr>
<td>private String EMPLOYEE_ID</td>
</tr>
<tr>
<td>private String LEVEL_ID</td>
</tr>
<tr>
<td>private String ITERATION_ID</td>
</tr>
<tr>
<td>private int C1</td>
</tr>
<tr>
<td>private int C2</td>
</tr>
<tr>
<td>private int C3</td>
</tr>
<tr>
<td>private int C4</td>
</tr>
<tr>
<td>private int C5</td>
</tr>
<tr>
<td>private int C6</td>
</tr>
<tr>
<td>private int C7</td>
</tr>
<tr>
<td>private int C8</td>
</tr>
<tr>
<td>private int C9</td>
</tr>
<tr>
<td>private int C10</td>
</tr>
<tr>
<td>private int C11</td>
</tr>
<tr>
<td>private int C12</td>
</tr>
<tr>
<td>private int C13</td>
</tr>
<tr>
<td>private int C14</td>
</tr>
<tr>
<td>private int C15</td>
</tr>
<tr>
<td>private int C16</td>
</tr>
<tr>
<td>public void Thread_checklist()</td>
</tr>
<tr>
<td>public void setTHREAD_ID()</td>
</tr>
<tr>
<td>public void setEMPLOYEE_ID()</td>
</tr>
<tr>
<td>public void setITERATION_ID()</td>
</tr>
<tr>
<td>public void setLEVEL_ID()</td>
</tr>
</tbody>
</table>

Figure 2.13. Thread_Checklist Class Diagram
public void setC1()
public void setC2()
public void setC3()
public void setC4()
public void setC5()
public void setC6()
public void setC7()
public void setC8()
public void setC9()
public void setC10()
public void setC11()
public void setC12()
public void setC13()
public void setC14()
public void setC15()
public void setC16()
public String getTHREAD_ID()
public String getEMPLOYEE_ID()
public String getITERATION_ID()
public String getLEVEL_ID()
public int getC1()
public int getC2()
public int getC3()
public int getC4()
public int getC5()
public int getC6()
public int getC7()
public int getC8()
public int getC9()
public int getC10()
public int getC11()
public int getC12()
public int getC13()
public int getC14()
public int getC15()
public int getC16()
public int insert()
public int delete()
public int update()

Figure 2.13 Thread_Checklist Class Diagram (cont.)
Project Checklist Class

The purpose of Project_Checklist class (Figure 2.14) is to provide delete, insert, update and select functions to manipulate the data in the Project_Checklist table.

<table>
<thead>
<tr>
<th>Project_Checklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>private String PROJECT_ID</td>
</tr>
<tr>
<td>private String EMPLOYEE_ID</td>
</tr>
<tr>
<td>private int C1</td>
</tr>
<tr>
<td>private int C2</td>
</tr>
<tr>
<td>private int C3</td>
</tr>
<tr>
<td>private int C4</td>
</tr>
<tr>
<td>private int C5</td>
</tr>
<tr>
<td>private int C6</td>
</tr>
<tr>
<td>private int C7</td>
</tr>
<tr>
<td>private int C8</td>
</tr>
<tr>
<td>private int C9</td>
</tr>
<tr>
<td>private int C10</td>
</tr>
<tr>
<td>private int C11</td>
</tr>
<tr>
<td>private int C12</td>
</tr>
<tr>
<td>private int C13</td>
</tr>
<tr>
<td>private int C14</td>
</tr>
<tr>
<td>private int C15</td>
</tr>
<tr>
<td>private int C16</td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>public void Project_checklist()</td>
</tr>
<tr>
<td>public void setPROJECT_ID()</td>
</tr>
<tr>
<td>public void setEMPLOYEE_ID()</td>
</tr>
<tr>
<td>public void setC1()</td>
</tr>
<tr>
<td>public void setC2()</td>
</tr>
<tr>
<td>public void setC3()</td>
</tr>
<tr>
<td>public void setC4()</td>
</tr>
<tr>
<td>public void setC5()</td>
</tr>
<tr>
<td>public void setC6()</td>
</tr>
<tr>
<td>public void setC7()</td>
</tr>
<tr>
<td>public void setC8()</td>
</tr>
<tr>
<td>public void setC9()</td>
</tr>
<tr>
<td>public void setC10()</td>
</tr>
<tr>
<td>public void setC11()</td>
</tr>
<tr>
<td>public void setC12()</td>
</tr>
<tr>
<td>public void setC13()</td>
</tr>
</tbody>
</table>

Figure 2.14. Project_Checklist Class Diagram
public void setC14()
public void setC15()
public void setC16()
public String getPROJECT_ID()
public String getEMPLOYEE_ID()
public int getC1()
public int getC2()
public int getC3()
public int getC4()
public int getC5()
public int getC6()
public int getC7()
public int getC8()
public int getC9()
public int getC10()
public int getC11()
public int getC12()
public int getC13()
public int getC14()
public int getC15()
public int getC16()
public int inert()
public int delete()
public int update()
UNIT TEST

Unit test is the initial step in the software testing phase. The unit test focuses on the smallest functional unit of design. This smallest functional unit can be single class, a program, internal procedures, or independent modules in an isolated test environment. In the RMT Tool, the unit test includes some of the following items to check each functional unit and ensure that all the command buttons work as expected.

- Check normal and abnormal program termination
- Verify the handling of all valid input and type
- Verify the handling of error conditions

While this is an enormous task, unit testing generally yields the highest number of detected problems within all testing techniques. A loose and sloppy unit test will cost someone a great deal of time in later stages of integration testing, system testing, and production usage. The results of the unit test for the RMT tool are listed in Table 3.1.
Table 3.1. The Results of the Unit Test.

<table>
<thead>
<tr>
<th>Forms</th>
<th>Tests Performed</th>
<th>Results</th>
</tr>
</thead>
</table>
| Authorization          | • Verify the handling of data type  
                          • Ensure Submit and Reset button work as expected | Pass    |
| Main Menu              | • Check the header label appearance  
                          • Ensure all links work as expected | Pass    |
| Create New Project     | • Verify the handling of data type  
                          • Check the header label appearance  
                          • Ensure Submit and Reset button work as expected | Pass    |
| Browse Current Project | • Check the choice box appearance  
                          • Verify the handling of selection  
                          • Ensure the link works as expected | Pass    |
| Organization           | • Check nodes position and drawing  
                          • Check nodes labeling appearance  
                          • Ensure image map works as expected | Pass    |
| Documentation          | • Check URL linkage  
                          • Check document panel appearance | Pass    |
| Thread Update          | • Verify the handling of data type  
                          • Check the header label appearance  
                          • Ensure Submit and Reset button work as expected | Pass    |
Table 3.1. The Results of the Unit Test. (cont.)

| Iteration Update | • Verify the handling of data type  
|                  | • Check the header label appearance  
|                  | • Ensure Submit and Reset button work as expected  
| Personnel Information Update | • Verify the handling of data type  
|                               | • Check the header label appearance  
|                               | • Ensure Submit and Reset button work as expected  
| Assign Personnel to Thread | • Check choice box items appearance  
|                           | • Selection  
|                           | • Check the header label appearance  
|                           | • Ensure Submit and Reset button work as expected  
| Multi Database Management | • Check the data that retrieved from database  
|                           | • Check the header label appearance  

Integration Test

The integration of the RMT Tool takes individual components and linking them with other components of the RMT Tool subsystems. The integrated system is the system to be tested. Please see Table 3.2 for detail.

47
Table 3.2. The Results of the Integration Test.

<table>
<thead>
<tr>
<th>Components Integrated</th>
<th>Test Performed</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>The RMT Tool forms and reports</td>
<td>Test the proper linkage between all the forms and report programs</td>
<td>Pass</td>
</tr>
<tr>
<td>The RMT Tool forms and their associated classes</td>
<td>Test the proper linkage between classes and the RMT Tool forms</td>
<td>Pass</td>
</tr>
</tbody>
</table>

System Test

After the Recursive Multi-Threaded Tool software is completely unit tested and integration tested, and meets all functional and performance requirements, the final step in the system validation process is the system test. Since the Algorithm 2000 Project and Algorithm 2001 Project, which are projects in the CSCI 455 (Software Engineering) course, were used for system testing. Data from the Algorithm 2000 and Algorithm 2001 were inputted into the RMT tool. The results from the system test provide actual information on how the RMT tool functions with the multi-database support.
CHAPTER FOUR
EXAMPLE SESSIONS

This project used JSP pages to build the user interface and used the JDBC classes to access the multi-database. This chapter shows how the JDBC classes created in this project support the RMT Software Process Management Tool.

Figure 4.1. The Main Structure of Demonstrate Window

As shown in Figure 4.1, there are two frames in the main window. The left frame is the menu page, which
contains a "Whole Project" link and names of the three projects. The right frame contains a project detailed page, which displays the detailed information of the selected link. All information brought up in this user interface is stored in the three databases.

By clicking on the link of whole project located in the left frame, the page will display the project information of these three projects in the right frame. There is also a project link (the plus sign) placed in front of each project name as shown in Figure 4.2.

Figure 4.2. Information Window for Whole Project
Clicking the project link, it will bring out the detail of the threads and sub-threads contained in the selected project. Figure 4.3 shows the structure image of the threads and sub-threads within the selected project. Exx%, which located after each phase name, stands for estimated progress, which is computed from the Check checkbox of the corresponding checklist page. Axx% stands for actual progress, which is generated from the Verify checkbox of the corresponding checklist page.

![Figure 4.3. Single Project View](image-url)
The project or thread name is also a link. Selecting a project or a thread will bring up the detailed information about that project or thread. In the right frame of the window shown in Figure 4.4, we see the information table for the selected thread.

Figure 4.4. Thread Information Page
Pushing the "Change Information" button located below the information table, the update page will be brought up in the right frame as shown in Figure 4.5. In this page, users can easily change the information for that selected project or thread.

Figure 4.5. Update Thread Information Page
As shown in the Figure 4.3, there is a Personnel link followed by the thread name. Clicking on this link, it will list all the team members who worked on that thread. As shown in Figure 4.6, the Thread Members table does not only list the team members' name but also the team leader's name and who is the management liaison.

![Thread Members Table]

Figure 4.6. Thread Members Page
The name link provided by Thread Members page will display the Personnel Information table of the selected member in the right frame (Figure 4.7).

![Personnel Information Table]

**Figure 4.7. Personnel Information Page**
Clicking on the check List link on the Thread Members page (Figure 4.6), the Checklist of Team Leader page will appear on the right frame of the window as shown in Figure 4.8. The Check checkbox is checked by the team leader and Verify checkbox is checked by the project leader.

Figure 4.8. Checklist Page
The overall progress is computed from two actual progress of two projects. The detailed information of two project is shown in Figure 4.9.

![Algorithm Progress Page](image)

Figure 4.9. Overall Progress Page
Clicking on the Specification phase of each thread or project located in any thread or project information page, will link to the software requirements specification of that thread or project. As an example, Figure 4.10 shows the Algorithm 2000 SRS - Prototype 2.

Figure 4.10. Specification Page
In any thread or project information page, the Planning phase of each thread or project is linked to the Software Project Management Plan of that thread or project, see Figure 4.11.

Figure 4.11. Planning Page
Through the Analysis phase link provided in any thread or project information page, users can obtain the analysis of that thread or project. Figure 4.12, shows the sequence diagram used in Algorithm 2000 in the right frame.

Figure 4.12. Analysis Page
Using the Design phase link of each thread or project placed in any thread or project information page, the design of that thread or project will be displayed in the right frame of the window, see Figure 4.13.

Figure 4.13. Design Page
Users can access the implementation information (Figure 4.14) of a thread or project by the Implementation phase link of each thread or project.

Figure 4.14. Implementation Page
Finally, this user interface includes the Testing phase link of each thread or project that allows the users to view the test cases used for the desired thread or project. Figure 4.15 shows an example of the testing page.

Figure 4.15. Testing Page
CHAPTER FIVE
MAINTENANCE MANUAL

Source Code

In the multi-database project, there are twelve HTML pages, twelve JSP pages, ten JAVA files and two SQL files. All the source files are stored in the attached disk. The /http directory contains all JSP and HTML pages. The sql statement files are stored in /sql directory. All JAVA files are put in the directory /java. The directory /document stores all word files.

Here is the list of the source files.

- Frame.htm
- Detail.jsp
- Project_menu.jsp
- Testdb.jsp
- WholeProject.jsp
- Personnel_detail.jsp
- proejctProp.jsp
- threadProp.jsp
- threadProp.jsp
- Sql4sqlServer.sql
- Sql4oracle.sql
Re-Compile

The source code can be re-compiled whenever there had been a change in any of the source files.

First of all, we need to specify the path where the command "javac" looks up the classes needed to run "javac," or referenced by other classes that is running "javac." If there are more than one directory, use semicolons to separate the directory names. It is often useful for the directory containing the source files to be on the class path. Moreover, users should always include the system classes at the end of the path. For example:

```
javac -classpath .;C:\users\dac\classes;
C:\tools\java\classes
```

Then change the directory to the one that holds the class or classes that need be to re-compiled. After changing the directory path, type the following command:

```
javac filename.java
```

Installation Process

There four major steps in the installation process. The first step is to store all the HTML and JSP files under the root directory of the Web server. Second, create a database instance in the database being used. Then start
the sql script. For Microsoft SQL server, run Sql4sqlServer.sql in Query Analyzer. As for Oracle database, run Sql4oracle.sql script in the sqlplus console. The final step is to store all the JDBC classes under the directory that was defined in the class path of the Web server.
CHAPTER SIX
CONCLUSION AND FUTURE DIRECTION

Currently, any organization that uses the relational database management system (RDBMS) probably runs multiple databases. Different databases may be associated with particular business functions, aligned with geographical boundaries, or accessed the same data in different ways, e.g., an order entry database whose transactions are aggregated and analyzed in a data warehouse. Usually, these databases are stored on different servers, which may be located at the same site or a continent away. With multi-database, data stored in multiple databases is accessible just as if it were stored in a single database.

Without this project, multi-database, the RMT software process management tool can only run on single server and use only one centralized database. With this multi-database, RMT not only can be run on different servers but also can be accessed by different software development teams located in geographically different locations.

There are three main functionalities provided by the multi-database project. One is to allow RMT users to create, delete, update and retrieve project, thread, personnel and iteration information from the database. The
second function is to support multiple sites showing only a single RMT hierarchical tree. In other words, an RMT project can store its information in different databases in different location. Once an RMT user wants to use data that is not stored in the local database, RMT can get the appropriate information from the corresponding remote site through the multi-database support. The third function is to let RMT link to the Personnel Software Process (PSP) method into the RMT process management tool. PSP can get more accurate data of a programmer’s working progress. We can get more correct working progress of threads or a whole project once we use PSP instead of checklists.

The multi-database architecture was developed mostly for RMT prototype. Therefore, it can be improved for further development of the RMT tool. For example, changing from multi-database to distributed database, which can allow the task of a project or a thread to be divided into different locations, where each location may use different kinds of databases. Another improvement is providing more bean classes for RMT interface to allow users to have more control through the interface. Furthermore, providing an auto install program, which will copy all files into the proper directory and run the suitable sql script automatically.
After implementing this Master Project, I have learned how to write JSP pages and become familiar with JSP. Indeed, I have obtained a lot of experience on installing and setting up of both Oracle database and Microsoft SQL Servers. I have also learned to create database link and snapshot. Since I used JRUN on the Web server, I have learned how to configure it. I can use Java Database Connectivity (JDBC) to access Oracle and MS SQL server database very well.
APPENDIX A

PSEUDO CODE
public class ConnPool
{
    set the default Max Connections to 2 ;
    set the Vector for freeConnections ;
    set the Hashtable for boundConnections ;

closeDB() or provides to free all object and close database
{
    if(boundConnections is not null){
        clear all the elements and close all the connections
        then set boundConnections to null ;   }
    if(freeConnections is not null){
        clear all the elements and close all the connections
        then set freeConnections to null ;   }
}

getConnection() gets connection from connection pool
{
    if(freeConnections is null)
        The connection pool has not been established yet. ;
    if(boundConnections still contain running threads)
        Cannot get connections over once for this current running thread ;
    if(freeConnections.size() is 0 )
        call wait() ;
}

openDB() opens database and setup connection pool
{
    set max connections to three ;
    for( i=0 ; i<maxConnections ; i++ )
        allows client to establish the connection
}

returnConnection() returns connection to connection pool
{
    if( conn is null )
        The connection which this current running thread got is not found ;
        The return the connection to pool
}

setConnectionSwitch():On is to open database . Off is to close database
{
    if(on_off.equalsIgnoreCase("ON") is true)
        call openDB() ;
    else if(on_off.equalsIgnoreCase("OFF") is true)
        call closeDB() ;
}
provide setMaxConnections() to set max connection number
provide setDriverName() to set driver name
provide setJdbcURL() url
provide setUserNam()e() to set username
provide setPassword() to set password
}

public class SQLBridge
{
    set ConnPool connPool; Connection conn; ResultSet rs; ResultSetMetaData rsmd; Statement stmt;

    in void clearResult() //clean all results
    {
        set all results and SQL statement to null;
    }

    in void closeDB() //return connections to connection pool
    {
        clearResult();
        if connPool is not null
        {
            connPool.returnConnection();
            connPool=null;
        }
        else
        {
            if( conn is null )
                This connection has been closed already.
            if( conn.isClosed() is true)
                This connection has been closed.
            conn.close();
        }
        conn=null;
    }

    in int execSQL() //execute SQL statement and return a integer
    {
        if( conn is null or conn.isClosed() is true )
            This connection has not been established yet.
        if( sqlStmt is null )
            SQL-statement is null;

    }

}
Call clearResult() to clear all result sets;  
Set conn.setAutoCommit() to true;  
set stmt = conn.createStatement();  
if( SQL statement starts with "SELECT" )  
{  
    set rs = execute SQL statement starts with "SELECT"  
    set rsmd = rs.getMetaData();  
}  
else  
{  
    set numRows = execute SQL statement which starts with "UPDATE",  
          "INSERT" or "DELETE";  
    call clearResult() to clear all result sets;  
    return numRows;  
}

in ResultSet execRecord()     // execute SQL statement and return a result set  
{  
    if( conn is null or conn.isClosed() is true)  
        This connection has not been established yet.  
    if( sqlStmt is null )  
        SQL-statement is null.;  
    Call clearResult() to clear all result sets;  
    Set conn.setAutoCommit() to true ;  
    Set stmt = conn.createStatement() ;  
    Set ResultSet rsTemp;  
    Set rsTemp = execute "select" SQL statement;  
    return rsTemp ;  
}

in void execUpdate()     //execute many SQL statements which start with  
                         "UPDATE", "INSERT" or "DELETE" in one function  
{  
    if( conn is null or conn.isClosed() is true)  
        The connection has not been established yet;  
    if( sqlStmts is null or sqlStmts.length is 0 )  
        SQL-statement is null;  
    Call clearResult() to clear all result sets;  
    Set conn.setAutoCommit( false ) to false;  
    for(int i is 0 ; i<total number of SQL statements ; i increase 1)  
    {  
        set stmt=conn.createStatement() ;  
        call stmt.executeUpdate( SQL statement) to execute SQL statement;  
        call stmt.close() ;  
    }  
    call conn.commit() to commit all transactions;  
}
in int getColumnCount()  //get the number of column
{
    if( rsmd is null )
        ResultSet is null.;
    return rsmd.getColumnCount();
}

in String[] getColumnNames()  //get column names
{
    if( rsmd is null )
        ResultSet is null.;
    return columnNames ;
}

in Object getField()  // get the field value with original type or convert to String
{
    if( rs is null or rsmd is null )
        ResultSet is null.;

    switch( rsmd.getColumnType(column) )
    {
        case Types is BIGINT :
            if( convertToString is true )
            return String.valueOf( rs.getLong(column) ) ;
            else
            return new Long( rs.getLong(column) ) ;

        case Types is BINARY :
            if( convertToString is true)
            return Byte.toString( rs.getByte(column) ) ;
            else
            return new Byte( rs.getByte(column) ) ;

        case Types is BIT :
            if( convertToString is true)
            return String.valueOf(rs.getBoolean(column)) ;
            else
            return new Boolean( rs.getBoolean(column) ) ;

        case Types is CHAR :
            return rs.getString(column) ;

        case Types is DATE :
            if( convertToString is true)
            return ( rs.getDate(column) ).toString() ;
            else
            return rs.getDate(column) ;
    }
case Types is DECIMAL:
  if (convertToString is true)
    return (rs.getBigDecimal(column, rsmd.getScale(column))).toString();
  else
    return rs.getBigDecimal(column, rsmd.getScale(column));

case Types is DOUBLE:
  if (convertToString is true)
    return String.valueOf(rs.getDouble(column));
  else
    return new Double(rs.getDouble(column));

case Types is FLOAT:
  if (convertToString is true)
    return String.valueOf(rs.getDouble(column));
  else
    return new Float(rs.getDouble(column));

case Types is INTEGER:
  if (convertToString is true)
    return String.valueOf(rs.getInt(column));
  else
    return new Integer(rs.getInt(column));

case Types is LONGVARBINARY:
  if (convertToString is true)
    return (rs.getBinaryStream(column)).toString();
  else
    return rs.getBinaryStream(column);

case Types is LONGVARCHAR:
  return rs.getString(column);

case Types is NULL:
  if (convertToString is true)
    return "NULL";
  else
    return null;

case Types is NUMERIC:
  if (convertToString is true)
    return (rs.getBigDecimal(column, rsmd.getScale(column))).toString();
  else
    return rs.getBigDecimal(column, rsmd.getScale(column));
case Types is REAL:
    if (convertToString is true)
        return String.valueOf(rs.getFloat(column));
    else
        return new Float(rs.getFloat(column));

case Types is SMALLINT:
    if (convertToString is true)
        return String.valueOf(rs.getShort(column));
    else
        return new Short(rs.getShort(column));

case Types is TIME:
    if (convertToString is true)
        return (rs.getTime(column)).toString();
    else
        return rs.getTime(column);

case Types is TIMESTAMP:
    if (convertToString is true)
        return (rs.getTimestamp(column)).toString();
    else
        return rs.getTimestamp(column);

case Types is TINYINT:
    if (convertToString is true)
        return String.valueOf(rs.getByte(column));
    else
        return new Byte(rs.getByte(column));

case Types is VARBINARY:
    if (convertToString is true)
        return (rs.getBytes(column)).toString();
    else
        return rs.getBytes(column);

case Types is VARCHAR:
    return rs.getString(column);

default:
    if (convertToString is true)
        return (rs.getObject(column)).toString();
    else
        return rs.getObject(column);
in Object getField()  //input the number of the column and get the column value and don't convert to String
{  
    return getField(column, false) ; 
}

in Object getField()  //input the name of the column and get the column value and don't convert to String
{  
    return getField(rs.findColumn(fieldName) ,false) ; 
}

in String getFieldString()  //input the number of the column and get the column value and convert it to String
{  
    return (String)getField( column, true ) ; 
}

in String getFieldString()  //input the name of the column and get the column value and convert it to String
{  
    return (String)getField(rs.findColumn(fieldName) ,true) ; 
}

in boolean nextRow()  //if rs is not null, return rs.next()
{  
    if( rs is null )  
        ResultSet is null.;  
        return rs.next() ;
}

int void openDB()  //open database
{  
    if( conn is not null and conn.isClosed() is not true )  
        The connection has been established already.;
    Call clearResult() to clear all result sets;
    Call Class.forName( drvName ) to set driver name;
    Set conn=DriverManager.getConnection( url ,uname ,passwd ) to get connection;
}

call void openDB()  //get connection from connection pool
{  
    if( conn is not null and conn.isClosed() is not true)  
        The connection has been established already.;  
    if( pool is null )  
        The connection pool cannot be found. ;
    Call clearResult() to clear all result sets;
Set connPool=pool;
Set conn=connPool.getConnection() to get connection form connection pool;
}

in void setConnectionSwitch() //To open or close database depend on user's input ("ON" or "OFF")
{
    if( on_off.equalsIgnoreCase( "ON" ) is true)
    {
        if( connPool is null )
            call openDB( driverName, jdbcURL, username, password ) to first time open database ;
        else
            call openDB( connPool ) to get connection from connection pool and database has already be opened;
    }
    else( on_off.equalsIgnoreCase( "OFF" ) is true)
        call closeDB() to close database or return connection to connection pool;
}

in void setErrMsg(errMsg)//set error message to errMsg
{
    set this.errMsg = errMsg;
}

in String getErrMsg () //get error message from errMsg
{
    return this.errMsg;
}

provide setConnPool() to set connection pool
provide setDriverName() to set driver name
provide setJdbcURL() url
provide setUserName() to set username
provide setPassword() to set password

public class Project extends SQLBridge
{
    //set value
    provide setPROJECT_ID() to set project id;
    provide setPROJECT_NAME() to set project name;
    provide setSTARTING_DATE () to set starting date;
    provide setESTIMATE_DURATION() to set estimate duration;
    provide setPROGRESS() to set progress;
provide setSPECIFICATION() to set specification;
provide setPLANNING() to set planning;
provide setANALYSIS() to set analysis;
provide setDESIGN() to set design;
provide setIMPLEMENTATION() to set implementation;
provide setTESTING() to set testing;

//get value
provide getPROJECT_ID() to get project id;
provide getPROJECT_NAME() to get project name;
provide getSTARTING_DATE() to get starting date;
provide getESTIMATE_DURATION() to get estimate duration;
provide getPROGRESS() to get progress;
provide getSPECIFICATION() to get specification;
provide getPLANNING() to get planning;
provide getANALYSIS() to get analysis;
provide getDESIGN() to get design;
provide getIMPLEMENTATION() to get implementation;
provide setTESTING() to get testing;

public int insert() //insert data into Project table
{
call super.execSQL(sql statement starts with "insert") to execute sql statement;
}

public int delete() //delete data from Project Table with two input parameters
{
call super.execSQL(sql statement starts with "delete" and contains one input constraints) to execute sql statement;
}

public int delete() //delete data from Project Table with four input parameters
{
call super.execSQL(sql statement starts with "delete" and contains two input constraints) to execute sql statement;
}

public int delete() //delete data from Project Table with six input parameters
{
call super.execSQL(sql statement starts with "delete" and contains three input constraints) to execute sql statement;
}

public ResultSet selectRecord() //retrieve data from Project Table with two input parameters
{
call super.execSQLRecord(sql statement starts with "select" and contains one input constraints) to execute sql statement;
}
public ResultSet selectRecord() //retrieve data from Project Table with four input parameters
{
    call super.execRecord(sql statement starts with "select" and contains two input constraints) to execute sql statement;
}

public ResultSet selectRecord() //retrieve data from Project Table with six input parameters
{
    call super.execRecord(sql statement starts with "select" and contains three input constraints) to execute sql statement;
}

public int update() //update data to Project Table with two input parameters
{
    call super.execRecord(sql statement starts with "update" and contains one input constraints) to execute sql statement;
}

public int update() //update data to Project Table with four input parameters
{
    call super.execRecord(sql statement starts with "update" and contains two input constraints) to execute sql statement;
}

public int update() //update data to Project Table with six input parameters
{
    call super.execRecord(sql statement starts with "update" and contains three input constraints) to execute sql statement;
}

public class Assignment extends SQLBridge
{
    //set value
    provide setPROJECT_ID() to set project id;
    provide setEMPLOYEE_ID() to set employee id;
    provide setTHREAD_ID() to set thread id;
    provide setPOSITION_ID() to set position id;

    //get value
    provide getPROJECT_ID() to get project id;
    provide getEMPLOYEE_ID() to get employee id;
    provide getTHREAD_ID() to get thread id;
    provide getPOSITION_ID() to get position id;
}
public int insert() //insert data into Assignment table
{
    call super.execSQL(sql statement starts with "insert") to execute sql statement;
}

public int delete() //delete data from Assignment Table with two input parameters
{
    call super.execSQL(sql statement starts with "delete" and contains one input constraints) to execute sql statement;
}

public int delete() //delete data from Assignment Table with four input parameters
{
    call super.execSQL(sql statement starts with "delete" and contains two input constraints) to execute sql statement;
}

public int delete() //delete data from Assignment Table with six input parameters
{
    call super.execSQL(sql statement starts with "delete" and contains three input constraints) to execute sql statement;
}

public ResultSet selectRecord() //retrieve data from Assignment Table with two input parameters
{
    call super.execSQL(sql statement starts with "select" and contains one input constraints) to execute sql statement;
}

public ResultSet selectRecord() //retrieve data from Assignment Table with four input parameters
{
    call super.execSQL(sql statement starts with "select" and contains two input constraints) to execute sql statement;
}

public ResultSet selectRecord() //retrieve data from Assignment Table with six input parameters
{
    call super.execSQL(sql statement starts with "select" and contains three input constraints) to execute sql statement;
}

public int update() //update data to Assignment Table with two input parameters
{
call super.execRecord(sql statement starts with "update" and contains one input constraints) to execute sql statement;
}

public int update()  //update data to Assignment Table with four input parameters
{
    call super.execRecord(sql statement starts with "update" and contains two input constraints) to execute sql statement;
}

public int update()  //update data to Assignment Table with six input parameters
{
    call super.execRecord(sql statement starts with "update" and contains three input constraints) to execute sql statement;
}

public class Personnel extends SQLBridge
{
    //set value
    provide setEMPLOYEE_ID() to set employee id;
    provide setFIRST_NAME() to set first name;
    provide setLAST_NAME() to set last name;
    provide setUSER_PASSWORD() to set password;
    provide setEmail() to set email;
    provide setPHONE1() to set phone1;
    provide setPHONE2() to set phone2;
    provide setPASS_PHRASE() to set pass phrase;

    //get value
    provide getEMPLOYEE_ID() to get employee id;
    provide getFIRST_NAME() to get first name;
    provide getLAST_NAME() to get last name;
    provide getUser_PASSWORD() to get password;
    provide getEmail() to get email;
    provide getPHONE1() to get phone1;
    provide getPHONE2() to get phone2;
    provide getPASS_PHRASE() to get pass phrase;

    public int insert()  //insert data into Personnel table
    {
        call super.execSQL(sql statement starts with "insert") to execute sql statement;
    }

    public int delete()  //delete data from Personnel Table with two input parameters
    {

call super.execSQL(sql statement starts with "delete" and contains one input constraints) to execute sql statement;
}

public int delete() //delete data from Personnel Table with four input parameters
{
call super.execSQL(sql statement starts with "delete" and contains two input constraints) to execute sql statement;
}

public int delete() //delete data from Personnel Table with six input parameters
{
call super.execSQL(sql statement starts with "delete" and contains three input constraints) to execute sql statement;
}

public ResultSet selectRecord() //retrieve data from Personnel Table with two input parameters
{
call super.execRecord(sql statement starts with "select" and contains one input constraints) to execute sql statement;
}

public ResultSet selectRecord() //retrieve data from Personnel Table with four input parameters
{
call super.execRecord(sql statement starts with "select" and contains two input constraints) to execute sql statement;
}

public ResultSet selectRecord() //retrieve data from Personnel Table with six input parameters
{
call super.execRecord(sql statement starts with "select" and contains three input constraints) to execute sql statement;
}

public int update() //update data to Personnel Table with two input parameters
{
call super.execRecord(sql statement starts with "update" and contains one input constraints) to execute sql statement;
}

public int update() //update data to Personnel Table with four input parameters
{
call super.execRecord(sql statement starts with "update" and contains two input constraints) to execute sql statement;
}
public int update() //update data to Personnel Table with six input parameters
{
    call super.execRecord(sql statement starts with "update" and contains three input constraints) to execute sql statement;
}

public boolean passCheck() //check user’s password
{
    super.execSQL(sql statement constrains with employee id and password);
    if (found the record in database)
        get FIRST_NAME from database;
    get LAST_NAME from database;
    return true;
    else //not found the record in database
        return false;
}

public class Thread extends SQLBridge
{
    //set value
    provide setTHREAD_ID() to set thread id;
    provide setPROJECT_ID() to set project id;
    provide setPROGRESS() to set progress;
    provide setTHREAD_NAME() to set thread name;
    provide setTOTAL ITERATION to set total iteration;
    provide setMOTHER_THREAD() to set mother thread;
    provide setSPECIFICATION() to set specification;
    provide setPLANNING() to set planning;
    provide setANALYSIS() to set analysis;
    provide setDESIGN() to set design;
    provide setIMPLEMENTATION() to set implementation;
}

//get value
provide getTHREAD_ID() to get thread id;
provide getPROJECT_ID() to get project id;
provide getPROGRESS() to get progress;
provide getTHREAD_NAME() to get thread name;
provide getTotal ITERATION to get total iteration;
provide getMOTHER THREAD() to get mother thread;
provide getSPECIFICATION() to get specification;
provide getPLANNING() to get planning;
provide getANALYSIS() to get analysis;
provide getDESIGN() to get design;
provide setIMPLEMENTATION() to set implementation;
provide getTESTING() to get testing;

public int insert()  //insert data into Thread table
{
  call super.execSQL(sql statement starts with "insert") to execute sql statement;
}

public int delete()  //delete data from Thread Table with two input parameters
{
  call super.execSQL(sql statement starts with "delete" and contains one input constraints) to execute sql statement;
}

public int delete()  //delete data from Thread Table with four input parameters
{
  call super.execSQL(sql statement starts with "delete" and contains two input constraints) to execute sql statement;
}

public int delete()  //delete data from Thread Table with six input parameters
{
  call super.execSQL(sql statement starts with "delete" and contains three input constraints) to execute sql statement;
}

public ResultSet selectRecord()  //retrieve data from Thread Table with two input parameters
{
  call super.execRecord(sql statement starts with "select" and contains one input constraints) to execute sql statement;
}

public ResultSet selectRecord()  //retrieve data from Thread Table with four input parameters
{
  call super.execRecord(sql statement starts with "select" and contains two input constraints) to execute sql statement;
}

public ResultSet selectRecord()  //retrieve data from Thread Table with six input parameters
{
  call super.execRecord(sql statement starts with "select" and contains three input constraints) to execute sql statement;
}

public int update()  //update data to Thread Table with two input parameters
{
call super.execRecord(sql statement starts with “update” and contains one input constraints) to execute sql statement;

public int update() //update data to Thread Table with four input parameters
{
    call super.execRecord(sql statement starts with “update” and contains two input constraints) to execute sql statement;
}

public int update() //update data to Thread Table with six input parameters
{
    call super.execRecord(sql statement starts with “update” and contains three input constraints) to execute sql statement;
}

public class Thread_archive extends SQLBridge
{
    //set value
    provide setTHREAD_ID() to set thread id;
    provide setPROJECT_ID() to set project id;
    provide setPROGRESS() to set progress;
    provide setEMPLEE_ID() to set employee id;
    provide setCURRENT_ITERATION() to set current iteration;
    provide setMOTHER_THREAD() to set mother thread;

    //get value
    provide getTHREAD_ID() to get thread id;
    provide getPROJECT_ID() to get project id;
    provide getPROGRESS() to get progress;
    provide getEMPLOYEE_ID() to get employee id;
    provide getCURRENT_ITERATION() to get current iteration;
    provide getMOTHER_THREAD() to get mother thread;

    public int insert() //insert data into Thread_archive table
    {
        call super.execSQL(sql statement starts with “insert”) to execute sql statement;
    }

    public int delete() //delete data from Thread_archive Table with two input parameters
    {
        call super.execSQL(sql statement starts with “delete” and contains one input constraints) to execute sql statement;
    }
}
public int delete() //delete data from Thread_archive Table with four input parameters
{
    call super.execSQL(sql statement starts with “delete” and contains two input constraints) to execute sql statement;
}

public int delete() //delete data from Thread_archive Table with six input parameters
{
    call super.execSQL(sql statement starts with “delete” and contains three input constraints) to execute sql statement;
}

public ResultSet selectRecord() //retrieve data from Thread_archive Table with two input parameters
{
    call super.execRecord(sql statement starts with “select” and contains one input constraints) to execute sql statement;
}

public ResultSet selectRecord() //retrieve data from Thread_archive Table with four input parameters
{
    call super.execRecord(sql statement starts with “select” and contains two input constraints) to execute sql statement;
}

public ResultSet selectRecord() //retrieve data from Thread_archive Table with six input parameters
{
    call super.execRecord(sql statement starts with “select” and contains three input constraints) to execute sql statement;
}

public int update() //update data to Thread_archive Table with two input parameters
{
    call super.execRecord(sql statement starts with “update” and contains one input constraints) to execute sql statement;
}

public int update() //update data to Thread_archive Table with four input parameters
{
    call super.execRecord(sql statement starts with “update” and contains two input constraints) to execute sql statement;
}
public int update() //update data to Thread_archive Table with six input parameters
{
    call super.execRecord(sql statement starts with “update” and contains three input constraints) to execute sql statement;
}

public class Position extends SQLBridge
{
    //set value
    provide setPOSITION_ID() to set position id;
    provide setDESCRIPTION() to set description;

    //get value
    provide getPOSITION_ID() to get position id;
    provide getDESCRIPTION() to get description;
    public int insert() //insert data into Position table
    {
        call super.execSQL(sql statement starts with “insert”) to execute sql statement;
    }

    public int delete() //delete data from Position Table with two input parameters
    {
        call super.execSQL(sql statement starts with “delete” and contains one input constraints) to execute sql statement;
    }

    public int delete() //delete data from Position Table with four input parameters
    {
        call super.execSQL(sql statement starts with “delete” and contains two input constraints) to execute sql statement;
    }

    public ResultSet selectRecord() //retrieve data from Position Table with two input parameters
    {
        call super.execRecord(sql statement starts with “select” and contains one input constraints) to execute sql statement;
    }

    public ResultSet selectRecord() //retrieve data from Position Table with four input parameters
    {
        call super.execRecord(sql statement starts with “select” and contains two input constraints) to execute sql statement;
    }
}
public int update() //update data to Position Table with two input parameters
{
    call super.execRecord(sql statement starts with "update" and contains one input
    constraints) to execute sql statement;
}

public int update() //update data to Position Table with four input parameters
{
    call super.execRecord(sql statement starts with "update" and contains two input
    constraints) to execute sql statement;
}

public class Iteration extends SQLBridge
{
    //set value
    provide setTHREAD_ID() to set thread id;
    provide setITERATION_ID() to set iteration id;
    provide setWEIGHT() to set weight;

    //get value
    provide getTHREAD_ID() to get thread id;
    provide getITERATION_ID() to get iteration id;
    provide getWEIGHT() to get weight;

    public int insert() //insert data into Iteration table
    {
        call super.execSQL(sql statement starts with "insert") to execute sql statement;
    }

    public int delete() //delete data from Iteration Table with two input parameters
    {
        call super.execSQL(sql statement starts with "delete" and contains one input
        constraints) to execute sql statement;
    }

    public int delete() //delete data from Iteration Table with four input parameters
    {
        call super.execSQL(sql statement starts with "delete" and contains two input
        constraints) to execute sql statement;
    }

    public int delete() //delete data from Iteration Table with six input parameters
    {
        call super.execSQL(sql statement starts with "delete" and contains three input
        constraints) to execute sql statement;
    }
}
public ResultSet selectRecord() //retrieve data from Iteration Table with two input parameters
{
    call super.execRecord(sql statement starts with "select" and contains one input constraints) to execute sql statement;
}

public ResultSet selectRecord() //retrieve data from Iteration Table with four input parameters
{
    call super.execRecord(sql statement starts with "select" and contains two input constraints) to execute sql statement;
}

public ResultSet selectRecord() //retrieve data from Iteration Table with six input parameters
{
    call super.execRecord(sql statement starts with "select" and contains three input constraints) to execute sql statement;
}

public int update() //update data to Iteration Table with two input parameters
{
    call super.execRecord(sql statement starts with "update" and contains one input constraints) to execute sql statement;
}

public int update() //update data to Iteration Table with four input parameters
{
    call super.execRecord(sql statement starts with "update" and contains two input constraints) to execute sql statement;
}

public int update() //update data to Iteration Table with six input parameters
{
    call super.execRecord(sql statement starts with "update" and contains three input constraints) to execute sql statement;
}

public class Iteration_phase extends SQLBridge
{
    //set value
    provide set ITERATION_ID() to set iteration id;
    provide set PHASE() to set phase;
}
provide setPROGRESS() to set progress;  
provide setDOCUMENT_URL() to set document url;

//get value  
provide getITERATION_ID() to get iteration id;  
provide getPHASE() to get phase;  
provide getPROGRESS() to get progress;  
provide getDOCUMENT_URL() to get document url;

public int insert() // insert data into Iteration_phase table  
{  
call super.execSQL(sql statement starts with "insert") to execute sql statement;
}

public int delete() // delete data from Iteration_phase Table with two input parameters  
{  
call super.execSQL(sql statement starts with "delete" and contains one input constraints) to execute sql statement;
}

delte(); // delete data from Iteration_phase Table with four input parameters  
{  
call super.execSQL(sql statement starts with "delete" and contains two input constraints) to execute sql statement;
}

delete(); // delete data from Iteration_phase Table with six input parameters  
{  
call super.execSQL(sql statement starts with "delete" and contains three input constraints) to execute sql statement;
}

public ResultSet selectRecord() // retrieve data from Iteration_phase Table with two input parameters  
{  
call super.execRecord(sql statement starts with "select" and contains one input constraints) to execute sql statement;
}

public ResultSet selectRecord() // retrieve data from Iteration_phase Table with four input parameters  
{  
call super.execRecord(sql statement starts with "select" and contains two input constraints) to execute sql statement;
}
public ResultSet selectRecord() //retrieve data from Iteration_phase Table with six input parameters
{
    call super.execRecord(sql statement starts with "select" and contains three input constraints) to execute sql statement;
}

public int update() //update data to Iteration_phase Table with two input parameters
{
    call super.execRecord(sql statement starts with "update" and contains one input constraints) to execute sql statement;
}

public int update() //update data to Iteration_phase Table with four input parameters
{
    call super.execRecord(sql statement starts with "update" and contains two input constraints) to execute sql statement;
}

public int update() //update data to Iteration_phase Table with six input parameters
{
    call super.execRecord(sql statement starts with "update" and contains three input constraints) to execute sql statement;
}
APPENDIX B

CHECK LIST
• Project Leader
  ■ Requirement Analysis
  System Environment
  Interface Definition
  Human Resource/Engineering Requirement
  ■ Planning
  Producing SQA
  Producing SPMP
  ■ Analysis
  Use Case Diagram
  Sequence Diagram
  Class Diagram
  Component Diagram
  Interaction Diagram
  State Diagram
  ■ Design
  Architectural Design
  ■ Implementation
  Distributing Object Modules to Team
  Collecting Object Modules from Team
  ■ Testing
  Team Level Unit Test
  System Wide Integration Test
• Team Leader
  ▪ Requirement Analysis
    Study and Understand SRS
    Specify Team Level Requirement
  ▪ Planning
    Study and Understand SQA
    Team Level Development Schedule Based on SPMP
    Organize Programmers Based on SPMP
  ▪ Analysis
    Decomposing Object
    Class Diagram
    Component Diagram
    Interaction Diagram
    State Diagram
  ▪ Design
    Number of Objects Completed
    Object Relation Definition
  ▪ Implementation
    Distributing Object Module to Programmer
    Collecting Object Modules from Programmer
  ▪ Testing
    Team Level Unit Test Done
Team Level Integration Test

- Programmer
  - Requirement Analysis
  - Study and Understand SRS and SQA
  - Specify Requirement for Object
- Planning
  - Programmer Level SQA
  - Programmer Level Development Schedule Based on SPMP
- Analysis
  - Modified Object
  - Delete Object
  - Added Object
  - Reused Object
- Design
  - Object Pseudo Code
  - Object Design - Number of Completed:
- Implementation
  - Modified Object
  - Delete Object
  - Added Object
  - Reused Object
- Testing
Object Unit Test

Programmer Level Integration Test
REFERENCES


