System for the Online Assessment of Distributed Projects

by

Fumiaki Shiraishi

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2004 [June 2004]

© Fumiaki Shiraishi, MMIV. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

Author .................................................................

Department of Electrical Engineering and Computer Science
May 18, 2004

Certified by ..... ..........................................................

Jonathon Cummings
Assistant Professor
Thesis Supervisor

Accepted by 

Arthur C. Smith
Chairman, Department Committee on Graduate Students
System for the Online Assessment of Distributed Projects

by

Fumiaki Shiraishi

Submitted to the Department of Electrical Engineering and Computer Science on May 18, 2004, in partial fulfillment of the requirements for the degree of Master of Engineering in Computer Science and Engineering

Abstract

The collaboration of researchers in different fields, institutions, and nations continues to be important as new ideas and solutions result from the combination of different knowledge and experiences. The Cambridge MIT Institute (CMI) Online Project Assessment System (OPAS) aims to collect information from projects funded by CMI and analyze the effects of geographic distance, project structure, and communication on performance. This thesis will discuss the development of the online data collection system as well as its maintenance and ongoing improvement.

Thesis Supervisor: Jonathon Cummings
Title: Assistant Professor
Acknowledgments

I would like to thank my supervisor, Professor Jonathon Cummings for the opportunity to work on this project. Though tough at times, he kept me on my toes making sure that progress was being made, on both the code as well as this thesis.

Furthermore, I would like to thank my parents for their support. Without them, I would not have had the opportunity to go through this Masters of Engineering Program. Finally, I want to thank my wife and chief editor, Mai, for dealing with me, my classes, my thesis, and all the stress.
Contents

1 Introduction ........................................... 13
   1.1 Introduction to Distributed Projects ................ 13
   1.2 Motivation for Online Assessment System ........... 14
   1.3 Objectives ....................................... 14
   1.4 System Overview .................................. 15
       1.4.1 Survey Subsystem ........................... 15
       1.4.2 Administrative Subsystem .................. 16
       1.4.3 Developer Subsystem ....................... 17
   1.5 Technical Background .............................. 18
       1.5.1 Terminology ................................ 18
       1.5.2 PHP ......................................... 18
       1.5.3 Apache, Mysql, HTML and Javascript ..... 21

2 Design and Development of the Survey Subsystem .... 23
   2.1 Overview ....................................... 23
   2.2 Functionality .................................... 24
   2.3 Design ........................................... 25
   2.4 Issues .......................................... 28
       2.4.1 Questions Requiring Special Tables ........ 29
       2.4.2 Saving the Results and Compaction of Data . 30
       2.4.3 User Identification and Uniqueness .......... 32
       2.4.4 Modularity .................................. 35
   2.5 User Experience Issues ............................ 35
2.5.1 Navigation ................................................. 35
2.5.2 Personnel Changes ................................. 37
2.5.3 Progress .................................................. 38
2.5.4 Summary Page ......................................... 38
2.5.5 User Feedback ........................................... 38
2.5.6 Tutorial and FAQ ................................. 39

3 Administration and Maintenance ......................... 41
3.1 Overview ................................................... 41
3.2 Design ...................................................... 42
3.3 Modules .................................................... 44
3.4 Data Issues ............................................... 46
  3.4.1 Excel Spreadsheet Structure .................. 46
  3.4.2 Special Characters and Data Formatting .... 46
  3.4.3 Checking for Human Error in the Data ....... 47
3.5 Maintenance Issues ..................................... 47
  3.5.1 Concurrent Code Fixes ......................... 47
  3.5.2 Concurrent Data Fixes ......................... 48
  3.5.3 Customer Service ................................. 49
  3.5.4 Bugs and Features ............................... 49

4 Conclusion .................................................... 51
4.1 Progress Update ......................................... 51
4.2 Lessons .................................................... 52
  4.2.1 Ramp-up Period .................................. 52
  4.2.2 User Interface .................................... 52
  4.2.3 Standards and Implementations ............ 53
4.3 Future Work .............................................. 54
  4.3.1 Maintenance, Small Features, and Bug Fixes 54
  4.3.2 Modularity and Cleanup ....................... 55
  4.3.3 Administrative and Analytical Pages ....... 55
List of Figures

1-1 Subsystems in OPAS .................................................. 16
1-2 PHP Function Pointer Example ........................................ 20
1-3 Variable Function Example ........................................... 20

2-1 Survey Flow Chart .................................................... 26
2-2 Code Flow in Response to User Input ................................. 27
2-3 Overall Structure of the Survey System ............................... 28
2-4 Layout of Navigation Elements ....................................... 36
2-5 Alternative Navigation System 1 ..................................... 37
2-6 Alternative Navigation System 2 ..................................... 37

A-1 index.php ............................................................... 61
A-2 person_info.php ....................................................... 62
A-3 tutorial.php ............................................................ 63
A-4 overview.php .......................................................... 64
A-5 survey.php; General Information Page ............................... 65
A-6 survey.php; CU Participants Page ................................... 66
A-7 survey.php; MIT Participants Page .................................. 67
A-8 survey.php; Industry Partners Page .................................. 68
A-9 survey.php; Participant Collaboration Page ......................... 69
A-10 survey.php; Organization of Work Page ............................ 70
A-11 survey.php; Communication and Travel Page ...................... 71
A-12 survey.php; Training and Outreach Page ........................... 72
A-13 survey.php; Ideas and Tools Page .................................. 73
List of Tables

2.1 Initial Results Table ........................................ 29
2.2 Final Results Tables .................................... 30
2.3 Shared Personnel Data Approach ....................... 31
2.4 Non-Shared Personnel Data Approach ................. 31
2.5 Two-tiered Approach ................................... 32
2.6 Initial User Table Example ............................ 33
2.7 Initial People Table Example .......................... 33
2.8 Final User Table Example ............................. 34
2.9 Final People Table Example ............................ 34

B.1 Tables in the Database Used by the Framework .... 84
B.2 Tables in the Database Used for the Questions ...... 84
B.3 Other Tables in the Database .......................... 84
B.4 PHP Files Used by the System ......................... 87
Chapter 1

Introduction

1.1 Introduction to Distributed Projects

Scientific researchers have collaborated with colleagues from different fields, backgrounds, cultures, and nations for many centuries. Many new ideas and breakthroughs have resulted from such collaborations in the past. By mixing expertise, backgrounds, and experiences, new approaches and ideas can be born. Collaborations, however, also face unique challenges which can hinder rather than help the research. In a previous study, Professor Cummings and his colleagues made observations based on a previous study of NSF funded projects[9]. In their research, they concluded that geographic dispersion had a significantly negative impact on performance. Another study by Professor Cummings and his colleagues have shown the effects of project structure on performance[8]. This study suggested that some project structures are better suited for group performance than others. With today’s technology and the wide assortment of communication tools available, collaboration should be easier and more successful than ever. Studies, however, have shown that this is not always the case. Technology does not always overcome distance[6], and communication does not always lead to successful working relationships[7].
1.2 Motivation for Online Assessment System

Cambridge-MIT Institute (CMI) is a strategic alliance between Cambridge University and Massachusetts Institute of Technology aimed at promoting intellectual exchanges and cross-boundary cooperations[2]. Specifically, CMI allows student exchanges and cooperative projects, where students and researchers of one institution can collaborate and form relationships with students and researchers from the other. CMI also provides masters and executive education programs, hosts conferences and workshops concerning competitiveness and entrepreneurship, and funds projects which promote exchanges between the two institutions and promote competitiveness in the UK.

The projects funded by CMI offer an unique opportunity where geographically dispersed projects can be studied. Most of the 63 projects which we are assessing are geographically dispersed and headed by co-principal investigators from MIT and Cambridge, respectively. From these projects, we should be able to observe the difficulties and possible solutions resulting from geographical dispersion.

Until recently, the assessment of these projects were made by using paper surveys. The answers to these paper surveys were collected into several spreadsheets. While these paper surveys were easy for the researchers to fill out, they provided only minimal information to the managers assessing these projects, and were also very difficult to collect and manage. An online system for collecting project information can greatly help the managers of CMI, allow more information to be collected, and perhaps allow simpler surveying process for the user in the long run.

1.3 Objectives

The CMI Online Project Assessment System (OPAS) has three main objectives. One objective is to simplify the collection of data from each of the CMI-funded projects, enabling better review and oversight. We are trying to improve on the process as well as the data collected. The second objective is to survey the investigators about the problems they faced concerning coordination and what solutions they attempted.
Studies of dispersed projects have often been inconclusive[10], partly because of the difficulty in collecting data. OPAS hopes to alleviate the problem by allowing researchers access to a greater data set. The final objective is to create a system that is robust and expandable. The CMI surveys will be taken repeatedly in the future, and changes to the questions, the projects, and the personnel are very likely. There is also the possibility of using this system to survey other groups of projects, which will most likely have their own set of questions as well as special requirements. Creating a system that can accommodate all these issues will be a big challenge.

1.4 System Overview

To go along with the three objectives, the system is built with three groups of users in mind. They are the researchers, who are asked to fill out the surveys and provide information, the administrators, who will be looking at the information collected by the surveys, and the developers who will be responsible for maintaining the system, fixing bugs, and adding new functionalities as the system matures. The researchers will only see what I will refer to as the Survey Subsystem, which is essentially an online application which will present the survey. The administrators will be looking at the Administrative Subsystem, which will display a variety of information such as survey progress, survey results, personnel information, etc... The developers will see all aspects of the system, but there will also be functionalities in the Administrative Subsystem which will be available only to Developers. These functionalities will include bug tracking, comments and feedback system, and logging system.

1.4.1 Survey Subsystem

The Survey Subsystem, which collects project information from researchers, is a simple web based application. After logging in, the user is asked to check their background information, go through a simple tutorial, then start filling out the survey forms. The survey form ranges from 4 to 18 pages in length, depending on the role of the researcher with respect to the project. A Principal Investigator, for example, will
be asked to answer 18 pages, with questions ranging from progress to cash-flow. An undergraduate researcher, on the other hand, will be asked about their own progress and their interaction with other researchers. These questions were decided on by Professor Cummings and the managers of CMI. Once the user has visited each and every survey page, they can move on to a summary page, where all the answers will be displayed. There is also a printer friendly version of the summary page, allowing simple printing.

The Survey Subsystem is currently being used by over 200 researchers, ranging from Professors to undergraduate students. While all users are assumed to be adept at the use of computers, the system still needs to be as self explanatory and user friendly as possible.

1.4.2 Administrative Subsystem

The Administrative Subsystem allows managers of projects to observe and review the progress of the surveying effort as well as the projects themselves. In the long run, this subsystem will probably see more use than the actual Survey Subsystem. Researchers will use the survey system only when they are required to, while CMI managers and administrators will use the administrative tools to make their daily tasks easier.
The Administrative Subsystem is expected to be used by ten or so users. While usability is still important, we can realistically ask the administrators to learn to use the system, and not all functionality need to be self-explanatory. Security and detailed permission setting, on the other hand, will be very important since every administrator would need a different setting depending on the role and task. Some administrators would be allowed to see only a portion of the projects, but will be allowed to update information, while other administrators will need to see all projects, but will only be able to observe the information.

1.4.3 Developer Subsystem

Currently the tools and facilities for developers exist together with the Administrative Subsystem. The developer subsystem works within the same framework, but is separated by access permissions. The key difference between the Administrative Pages and Development Pages is the content which they monitor. The Administrative Pages focus on the projects, the people, and the survey answers. The Developer Pages focus on the system itself rather than the data. The Developer Pages will record information on access records, platforms and browsers used by people, and bug reports. The Developer Pages will also allow developers to debug and fix minor problems using simple tools.

While tools are important, in the end the developers will need to fix the code or the database directly. For developers, clean modular code, concise comments, and straightforward database design is much more valuable than fancy tools. This paper does not have a chapter dedicated to developer systems because the entire system is written with developers in mind. Instead, each section will discuss modularity and scalability, which are issues based more on developer needs rather than user needs.
1.5 Technical Background

1.5.1 Terminology

To simplify the understanding of this thesis, the “client” will refer to the HTML browser. Any action on the client side are actions performed by the user on a browser such as Internet Explorer or Netscape. A client side action handling are error checks and state changes that are performed purely in the browser without any information passed to the web server. On the other hand, the “server side” will always refer to the web server which also runs the PHP scripts. While the database is also a separate server, this part will simply be called the “database”, while the term “server” will be reserved for the HTML server.

The word “user” will refer to any of the researchers who are asked to use this system and fill out a survey. The word “PI” on the other hand, stands for Principal Investigator, and are leaders of their respective projects. The word “PI” will appear frequently because they are the main target for this system, and also have additional pages to fill out compared to other research participants. The word “administrators” will refer to managers from the CMI office who will not be filling out surveys, but will be active in monitoring the projects and are interested in observing the information resulting from the survey effort. Finally, “developers” will include those who are directly involved in the development and maintenance of the system.

1.5.2 PHP

PHP is a simple script language designed specifically for the implementation of dynamic web pages[4]. This language enables easy and rapid development of online web applications. PHP is an open source language with no corporate support, but with wide community backing. While no one is responsible for the support of PHP, documentation and information is plentiful, and functions exist for all sorts of purposes. Consequently, PHP comes equipped with a full set of SQL functions, simplifying the use of databases with these dynamic web pages. The combination of PHP and the
MySQL server is very common and well documented. This combination is used to implement this online system as well.

There are two characteristics to know about PHP that have significant implications. First, PHP is a purely server side language. PHP can create HTML pages dynamically as a result of an HTTP query, but PHP cannot perform actions on the client-side. As a result, PHP is limited in what it can do for user interface. Unlike Java applets or normal compiled programs, PHP has limited and indirect control over the user interface. Any client-side action handling, such as error checking, animations, and effects, must be handled using Javascript. The result is that PHP must often dynamically create a Javascript function, which can then handle client-side effects.

The second note about PHP, which is directly related to the first, is that PHP is not used to write a program that can be used by the end users. Rather, PHP is used to create a program, made up of HTML and Javascript, which is used by the end user. Because PHP is pretty simple to begin with, this level of indirection is usually not a problem. There are, however, instances which should not occur in a compiled language. In the survey system, for example, a different Javascript function must be called on the browser depending on what page the user is and what the user does. To make this possible, the PHP passes a pointer to a function which dynamically creates a Javascript function, and this pointer is eventually called to place the Javascript function in the proper place.

One interesting feature of PHP that is not found in compiled languages such as Java or C is the concept of variable functions. In PHP, a string variable can be set to a function name with arguments, and then called like a normal function. While this feature is not necessary in most instances, it can be used to simplify modularity. For example, by using variable functions, PHP easily allows dynamic loading and unloading of code from a database. A developer could register code modules in a database table for use by some framework. The framework, could then query the database, select the appropriate modules, and run them. Similar implementations are possible with Java and C as well, but will require the use of dynamically linked functions or overlays, and will be considerably more complex.
function writeJavascript()
{
    // Set $string as the string which will define the javascript function "reconfirm()"
    $string = "<script language='javascript'>
    function reconfirm(form)
    {
        if (confirm('You sure?'))
        return true;
    
    return false;
    }
    
    </script>";

    // Write out the string to create the html result
    echo $string;
}

Figure 1-2: PHP Function Pointer Example

$foo = "myFunction()";
$foo; // calls myFunction();

Figure 1-3: Variable Function Example
1.5.3 Apache, Mysql, HTML and Javascript

The other technologies used in the implementation are the Apache web server, MySQL database server, HTML and Javascript. While these are commonly used technologies, I will quickly describe each one, and what the benefits and costs were.

Apache web server is the open source web server that has been around since 1995[1]. It is being developed and maintained by the Apache group, who has now grown to become an open source community that is developing much more than just this server. The Apache server is a good choice because it performs reasonably well compared to proprietary web servers, documentation is abundant, and the server is also very stable. Furthermore, there are many plug-ins and tools that work well with Apache and are also free, such as PHP.

MySQL database is the self-proclaimed “The World’s Most Popular Open Source Database” and is developed and maintained by the MySQL group[3]. Though MySQL is not known for exceptional performance, and though there are many other open source databases available, MySQL has recently seen a surge in popularity mainly because of its synergy with PHP. Development using PHP and MySQL is extremely simple and also well documented on the 1.

HTML and Javascript are the de facto interface on the Internet. While the implementation of HTML and Javascript differ depending on the browser, with care one can write an application that can be accessed and used anywhere on the web. PHP, Apache, and MySQL are all used to simply output HTML pages which the user can use.
Chapter 2

Design and Development of the Survey Subsystem

2.1 Overview

The online survey system is intended to allow academic researchers and their collaborators to enter information regarding their project in an easy and organized fashion. While I assumed that most users would be very comfortable with computers, usability was still an important issue. Online surveying and questioning would be seen as a distraction to researchers, and many will try to avoid or minimize the effort required for the survey. Good user experience is crucial in assuring participation and quality of data.

There were also two other factors which also drove the development of the system. One is the ability to easily modify or update the questions involved in the survey. The other factor is the general presentation and the professional look of the system. This chapter will talk about the design and implementation of this system, as well as some of the main issues during development.
2.2 Functionality

The flow of the survey process is designed with user friendliness in mind. There is a default flow that a first time user can take with little thought or effort. For experienced users, however, they have the option to choose and work only on what matters to them. The following is a brief explanation of each step in the survey process.

- **User Login and Initialization** The first task the system must do is verify the user. The username and password is checked with the database for accuracy. If there is a match, the user’s information is retrieved and the user is asked to verify their background information.

- **Project Selection and Survey Initialization** If the user is involved in more than one project, the user is asked to choose which project they would like to work on. If the user is involved in only one project, that project is selected automatically and the project selection page is skipped. If the user has answered information regarding this project before, then their previous answers are retrieved and the user is directed to the page where they last left off. If this is the first time the user is working on this project, the user is taken to the Overview Page. The user will also be taken to the Overview Page if the entire survey has already been filled out, and the user is returning to modify their previous answers.

- **Tutorial Page** Before the user is shown to the actual survey pages, they are first shown the Tutorial Page. This page shows each page in the survey, why the questions are asked, and what kinds of answers are expected. In addition, there is also a Frequently Asked Questions section at the bottom of the tutorial page. This page was added after the initial launch to try to eliminate some of the confusion the users had.

- **Overview Page** This page is shown only to users who are working on this project for the first time or have already filled out the survey for this project.
For users logging in for the first time, the Overview page will show all the sections that the user will be asked to fill out, but will not let the user select or unselect any of the sections. First time users, by policy, must view all pages at least once. For users who have already filled out all of the survey, this page will allow them to select which sections they would like to review and modify.

- **Survey Pages** The survey pages contain the actual questions that are asked to the researchers regarding their project. The questions that are asked and the processing of these questions are handled in each respective question module. One question module is designed to deal with one question type. A question type can be a simple text-box style question to the more complex milestone progress question.

- **Summary Pages** Once the user has viewed all the pages, they can move on to the Summary page. If they have not done so already, they are asked to verify their responses here. There is also a printer friendly version of the Summary Page, which allows the researchers to print out all their answers.

### 2.3 Design

The basic design of the Survey Subsystem is straight forward. The HTTP connection with the client browser is handled by the Apache Web Server. The HTTP request then calls the appropriate PHP script to interact with the MySQL database. The resulting HTML page from the PHP is then sent back to the client by the Apache Web Server. The overall structure is displayed in Figure 2-3.

The code and the database tables can be roughly divided into two categories. One category is the core framework. The core framework allows for the basic functionalities of an online survey, and is expected to be reused without modifications for any other online surveys we might do. Basic functionalities include user authentication and initialization, displaying user background information, basic survey question handling, displaying of summaries, navigation, and logging out. The other category is the
Figure 2-1: Survey Flow Chart
survey specific questions. The code and the database tables in this category allows the system to handle questions like personnel, check box questions, budgets, and milestones. This part of the code is expected to be project specific, and is not expected to be reused without modification. Figure 2-2 shows an example of code flow in response to user input in the survey pages. The figure shows which parts of the code are considered to be part of the framework and which parts are part of the survey specific code.

The two important PHP scripts in the framework are survey.php and submit.php. All HTML generation is processed in survey.php. The survey.php will call the appro-
appropriate functions and generate an HTML output, depending on what page the user is on, and what the user’s survey answers are. When a user submits data, the submit.php script is called. The submit.php parses all the fields submitted by the user, arranges them in order, and calls all the appropriate functions to communicate with the database.

Each question type has its own file, such as cashflow_func.php which contains the functions for the budget page, and personnel_func.php which contains all the functions for dealing with research personnel. In these question specific files are HTML generating functions called by survey.php, and database handling functions called by submit.php.

2.4 Issues

The original version of the survey system was very straightforward. Every question had a column in the results table, along with a function which handled the interface, a function which handled the packing and unpacking of the data, and a function which handled the display in the summary page. The advantage to this approach would have been conceptual clarity and the ability to add new question types with only a column addition to the database table. This scheme, however, quickly turned out to be inadequate because of all the special cases and special user interface needs.
Initially, I had decided that all answers from a user were going to be placed in one database table named “results”. An entry in this results table would contain one field for every question in the survey. Table 2.1 shows an example of this table structure.

A textbox question would simply place the text in the results. The budget question, on the other hand, would take all the budget estimates, encode it into one string delimited by special characters, and placed in one field. Formatting and packaging of data was done for every type of question. This scheme, however, was not enough for the kinds of questions we wanted in the survey.

The first problem was personnel data. PIs were asked to verify information about every participant in the project. Packaging the information about every participant in the project into the results table seemed redundant, considering that personnel already had its own dedicated table for managing personnel and user accounts. Personnel information was decided to be a special case which does not have a field in the results table, but deal directly with the people table, and the people_activity table, which I will talk about later.

The second problem was budget data. While packaging budget data into the results table would not have been difficult, I felt that budgetary information would be used for various statistical and accounting purposes. If I were to package all these numbers into a string, I would also have to parse and unpack these numbers for every calculation or analysis I would need to do in the future. Instead, I created a table specifically for budget information, which would allow for easy calculation and accounting.

The final exception was milestone data. Similar to budget information, packaging the milestone data into the results table would not have been difficult. However, I felt
that there might be a need for more project management and scheduling analysis in
the future. In such a case, storing the milestone information in an easy to understand
form would be valuable in the long run.

The final database design, shown in Table 2.2, is quite different from what I had
initially. The database is significantly more complex, and the handling of data is
full of special cases. The benefit, however, is the conceptual clarity of each database
table. As can be seen in the example, the information in the people table, the budget
table, and the milestone table is obvious just by examining them directly. The tables
can be directly exported to Excel and will make sense to humans. No unpackaging
tool are needed.

2.4.2 Saving the Results and Compaction of Data

As a result of the issues I mentioned in the previous special tables section, I created
a people table, a budget table, and a milestone table. These tables were originally
shared by all researchers in a project. Table 2.3 shows an example of this method
for project personnel data. If a user modifies a figure in the budget table, this
<table>
<thead>
<tr>
<th>projectID</th>
<th>perceiver's personID</th>
<th>personID</th>
<th>name</th>
<th>active/inactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>100</td>
<td>Bob Jones</td>
<td>active</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>101</td>
<td>Mary Smith</td>
<td>active</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>100</td>
<td>Bob Jones</td>
<td>inactive</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>101</td>
<td>Mary Smith</td>
<td>active</td>
</tr>
<tr>
<td>2</td>
<td>103</td>
<td>102</td>
<td>Joe Jones</td>
<td>inactive</td>
</tr>
<tr>
<td>2</td>
<td>103</td>
<td>103</td>
<td>Jane Jackson</td>
<td>active</td>
</tr>
</tbody>
</table>

Table 2.3: Shared Personnel Data Approach

<table>
<thead>
<tr>
<th>projectID</th>
<th>perceiver's personID</th>
<th>personID</th>
<th>name</th>
<th>active/inactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>100</td>
<td>Bob Jones</td>
<td>active</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>101</td>
<td>Mary Smith</td>
<td>active</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
<td>100</td>
<td>Bob Jones</td>
<td>inactive</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
<td>101</td>
<td>Mary Smith</td>
<td>active</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>102</td>
<td>Joe Jones</td>
<td>active</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>103</td>
<td>Jane Jackson</td>
<td>inactive</td>
</tr>
<tr>
<td>2</td>
<td>103</td>
<td>102</td>
<td>Joe Jones</td>
<td>inactive</td>
</tr>
<tr>
<td>2</td>
<td>103</td>
<td>103</td>
<td>Jane Jackson</td>
<td>active</td>
</tr>
</tbody>
</table>

Table 2.4: Non-Shared Personnel Data Approach

Modification would be seen by all other researchers in the project. This, however, turned out to be a mistake, and each researcher would need their own copy of the results.

Table 2.4 is an example of the non-shared personnel data. Each researcher (perceiver) has a list of every member in his project, and status for each member. This allows the possibility of different researchers having different perceptions of their project. In the example, Joe Jones (personID 102) and Jane Jackson (personID 103) have different answers regarding personnel activity.

Having a complete set of project data for each participant, however, would have been extremely inefficient in terms of memory use. For example, if we were to survey 50 projects, each with 20 researchers, we would need 1000 sets of project data, or 20,000 entries in the personnel table! To reduce the number of table entries, I constructed a two-tiered approach. For each project, there was one set of master data. This master data would contain the initial values for personnel, budgets, and milestones. Whenever a researcher modifies these values, a dedicated copy is made for this researcher, and that new value is stored in the activity table. In the worst case where everyone modifies every data, the memory use would still be the same as the
### Personnel Table

<table>
<thead>
<tr>
<th>projectID</th>
<th>personID</th>
<th>name</th>
<th>active/inactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>Bob Jones</td>
<td>active</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
<td>Mary Smith</td>
<td>active</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>Joe Jones</td>
<td>inactive</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>Jane Jackson</td>
<td>active</td>
</tr>
</tbody>
</table>

### Activity Table

<table>
<thead>
<tr>
<th>projectID</th>
<th>perceiverID</th>
<th>personID</th>
<th>name</th>
<th>active/inactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101</td>
<td>100</td>
<td>Bob Jones</td>
<td>inactive</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>102</td>
<td>Joe Jones</td>
<td>active</td>
</tr>
</tbody>
</table>

Table 2.5: Two-tiered Approach

non-shared approach. In practice, however, the data would be significantly smaller since we are only saving data that someone has deliberately changed. Table 2.5 shows an example of this two-tiered approach, where the personnel table contains all the data, and the activity table shows only the modifications.

As mentioned earlier, the system is currently assessing 63 projects. Based on those 63 projects and their respective researchers, the non-shared approach would have required 5511 entries in the table. Using the two-tiered approach, however, the system currently has 249 entries in the personnel table and 92 entries in the activity table, resulting in 341 entries total.

### 2.4.3 User Identification and Uniqueness

The initial implementation of the system used two tables to maintain information regarding personnel, the people table and the user table. Initially, the people table contained contact information, email address, first name, last name, and middle name. The user table, on the other hand, contained system related information such as user name, password, permission level, and project ID. Tables 2.6 and 2.7 show an example of the first user and people table structure.

When I finally received the data to populate the tables and prepare for launch, I realized that the current scheme was severely flawed. The main problem was the possibility of one person being involved in more than one project and having a dif-


<table>
<thead>
<tr>
<th>uID</th>
<th>projectID</th>
<th>permissionLevel</th>
<th>personID</th>
<th>userName</th>
<th>password</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>100</td>
<td>jones</td>
<td>1234abcd</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>101</td>
<td>smith</td>
<td>abcd1234</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>102</td>
<td>jones</td>
<td>ab12ab12</td>
</tr>
</tbody>
</table>

Table 2.6: Initial User Table Example

<table>
<thead>
<tr>
<th>personID</th>
<th>projectID</th>
<th>uID</th>
<th>name</th>
<th>email</th>
<th>etc...</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>1</td>
<td>Bob Jones</td>
<td><a href="mailto:bob@some.email">bob@some.email</a></td>
<td>...</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>2</td>
<td>Mary Smith</td>
<td><a href="mailto:mary@some.email">mary@some.email</a></td>
<td>...</td>
</tr>
<tr>
<td>102</td>
<td>2</td>
<td>3</td>
<td>Joe Jones</td>
<td><a href="mailto:joe@some.email">joe@some.email</a></td>
<td>...</td>
</tr>
</tbody>
</table>

Table 2.7: Initial People Table Example

Different permission level for each project. For example, one professor was the Principal Investigator for one project, but also a participating researcher in another. While I could have added some functionality to the previous layout to cope with this new problem, the solution would have been inelegant, and difficult to maintain in the long run.

Another problem that I discovered was how a lot of the information was unreliable. Originally, I had used the email address as the indexing field to sort and differentiate people. I knew that email addresses would be unique, and that no two researchers would have the same email address. When I started receiving actual data, however, I discovered that while no two researchers shared the same email address, one researcher often had many different email addresses. To make matters worse, email addresses changed often, making email an unreliable index. In fact, there was no reliable index. Last name and first name often conflicted or were spelled wrong.

Though this was only a week before the final launch, I was forced to rewrite much of how personnel information was organized and stored. The result, however, is much more robust and easy to maintain than what I had earlier.

The final configuration again uses two tables, the people table and the user table, as can be seen in tables 2.8 and 2.9. This time, the user table contains all information that is direct to the person and is project independent. First name, last name, and contact information were placed in the user table, as well as title, department, email address, and personal web page URL. The people table, on the other hand, contains
personnel information relevant to a specific project. The project position information and permission level were placed in this people table. Now, all researchers would have just one entry in the user table, but could have multiple entries in the people table, depending on how many projects they were involved in.

To make this new structure possible, I came up with a unique tag system. Whenever a person is added to the system, the person is given a unique tag. The tag would be placed in the user entry as well as the people entry, allowing cross referencing. To make debugging simpler, the unique tag is the person’s last name followed by a number. When two people have the same last name, the number at the end would be incremented to make the tags unique. In theory, however, the unique tags could be anything as long as they were unique. The last names were used, just to allow manually debugging. Giving random numbers after people’s names might seem strange, but I felt this was acceptable since users would never actually see these tags. The users would only see their username and password.

In traditional systems, the usernames could have been used as unique tags as well. In our system, however, we had decided early on that usernames could not be unique. If there were two researchers with the last name “Jones”, we did not want one user to have “jones” and the other as “jones1” for the username. While I doubted that any professor would take offense at being given a random number following their name, I did not want to tempt fate, knowing that we would have plenty of objections to an online survey system anyway.
2.4.4 Modularity

Modularity of this system is important for future maintenance and expansion. Specifically, there are two possibilities in the future where modularity would be important. The first is the possibility of modifying, removing, or adding a new question type to the survey. The second is the possibility of adapting the entire system to run surveys for an organization other than CMI.

As a result, two sets of boundaries will be needed in the code. First, the code for each question type must be separate from the code for the main framework as well as from other question type. This will simplify the addition or subtraction of a question type, as well as reduce the chance of side effects when modifications to the question are made. Secondly, the code relating to flow and data must be separate from the code that manages the interface and the graphics. Ideally, adapting the system for a new organization will require modifying only a few files to alter the look and feel of the interface.

2.5 User Experience Issues

2.5.1 Navigation

The final implementation of the survey offers two modes of navigation. One is the page tabs towards the bottom of the page. These tabs show what page the user is currently on and allows the user to select any other page to jump to. In addition, the tabs will show how many of the pages have already been visited and how many are still left by changing colors. The other form of navigation is the “next” link at the bottom right of the page. This link will take the user to the next unvisited page. If all pages have already been visited, the “next” link will change to “summary” link and take the user to the summary page.

The benefits of this navigation system are compactness and completeness. The tabs and the link take very little space, and allows the entire page to fit in one computer screen. The tabs allow the user to visit any page while the "next" link
easily takes the user to the next necessary page. Finally, the tabs are colorful and look professional, something that was a major concern for the CMI office.

The possible flaws with the navigation system are redundancy and lack of content information. In theory, the “next” link is not necessary until the very end. The tabs allows the user to select every page, and we just needed another link to appear when the user is ready to go to the “summary” page. In addition, the “next” link is potentially a source of confusion. Rather than simply taking the user to the next page, the “next” link takes the user to the next unvisited page. For example, the user could be on page 10, but the “next” button could take the user to page 9 if that page has not yet been visited. This was done to ensure that the user visits all pages, but could potentially lead to confusion. Furthermore, the “next” link changing into the “summary” link might also be a source of confusion. The problems with giving one button or link different functionalities based on context is well documented[11]. Finally, the current navigation system tells very little about what each page contains. If the user were to place the mouse over the tab, a small pop-up window would appear with the name of the page. However, it is very unlikely that any user would notice this feature.

One possible alternative would have been to banish the page tabs and simply have “back” and “next” buttons, as can be seen in Figure 2-5. This would be simpler and would be consistent with the web model of navigation. We could also have a simple display to indicate the user’s progress, like “4 out of 10.” A problem with this approach would have been a lack of color and professional feel to the page. This
Another possibility would be to display a vertical index to the left of the screen instead of the page tabs, as in Figure 2-6. By having a vertical index, we could display the name for each page, possibly even a description. The cost for this option would be screen space. This system would have taken much more space compared to the page tabs. Such an index also tends to have a big unused space below it, when the page becomes very long.

2.5.2 Personnel Changes

One feature that required much time and effort was the updating of personnel information. Whenever a new researcher is added to a project, that researcher is sent an email of invitation. Because this email is sent, we must be sure that the information
we were given is correct. We have three level of checks for this. First, when a new researcher information is filled and submitted, the server will check the fields for obvious mistakes. The email address, for instance, will be checked to see if the text is formatted correctly. Second, the user entering the new researcher is then asked if the information is correct and that this person should be added to the system. Finally, once the user is entered, an administrator could check the new entry before sending an email.

2.5.3 Progress

Another feature we added was the recording of user progress. We assumed that most users would enter the survey in several sittings. A problem with this is the amount of overhead required to return to where they left off. To ameliorate this, we keep track of what the user has done. When a user returns to the survey, the user is automatically taken to the next page the user needs to visit so that they can immediately start on the survey where they left off.

2.5.4 Summary Page

With most online systems, there is always the fear that any information filled is not being stored properly. For this, we have a summary page, where the user can review what they have entered and also check for accuracy. There is also a printer-friendly version, allowing the user to print out their answers.

2.5.5 User Feedback

Our project lacked the man power and time for thorough testing of all functions in the system. As a result, feedback and bug reports from users were both important and unavoidable. Initially, we offered a comments field at the bottom of every page as well as an email address to which users could voice their concern. When this proved insufficient, we also implemented a bug reporting system, where a user could simply explain what happened and what went wrong. The bug reporting system was
different from the comments field in that it also recorded the platform and browser information for that user, as well as all state variables. While we could not be sure that the user would report their bug immediately after it occurs, or even if they would report from the same machine, we hoped that this information might come in handy.

2.5.6 Tutorial and FAQ

Much of the feedback from the users were about what they were suppose to do and why. With some survey questions, the users did not understand the point or the purpose, while with other survey questions the users were simply confused about what was being asked. The users did not know how many questions there were total, and where they were suppose to provide information. These were serious problems, because most users seemed to lose motivation and any willingness to cooperate after seeing a few survey pages.

We decided we needed two things to improve the survey process. First, we needed a road map which would prepare the user for each question and what to expect next. By letting the user see the entire survey, the user would know what they will have to do, and where they can provide the information regarding their project. Second, we needed a way to explain why each question was being asked. Many of the researchers were unused to the kinds of questions on the survey, and seemed to dismiss these questions as meaningless or even a waste of time. By explaining why each question is there and what kinds of answers we expect, we tried to provide a motivation for the question, and hence give a reason for the user to cooperate. With regards to how to use the page, and answer the question, we decided that we simply needed better explanation text on each page.

We did not want to add more information to the survey pages, since they were already very crowded. Furthermore, I wanted to keep the tutorial as simple as possible, so that we would not need a tutorial to navigate the tutorial. The result is the current Tutorial Page, which is a simple one page display which simply combines screen shots and text to set up expectations for the user and provide a motivation for each question. The FAQ was added at the bottom of the Tutorial Page to specifically
answer a lot of the questions that were asked by the users.
Chapter 3

Administration and Maintenance

3.1 Overview

Immediately after launch, I was the sole developer, and was responsible for most data changes, bug fixes, and feature additions. While I wrote several makeshift scripts to simplify the process of modifying or fixing the data, much of the work was done directly to the database using phpMyAdmin, a database management tool[5]. This approach was very time consuming and was obviously not maintainable in the long run. With the Adminive Subsystem, the initial goal was to first simplify the task of monitoring and editing the data, so that eventually these tasks could be handed over to others.

Initially, I knew very little of what I would need for an Administrative Subsystem. I had some guesses as to what kinds of functionalities we would need, but not a complete picture. Fortunately, I had started on the Admin Subsystem after the Survey Subsystem, and I was much more comfortable with PHP. I had a better idea of the strengths and weaknesses of the language, as well as how I should design the system as a whole.

I decided that the two important features of the admin pages were going to be fast development time, and a common framework that was flexible enough that I could add any kind of administrative modules that I would need later.
3.2 Design

Compared to the Survey Subsystem, the Admin Subsystem has a very minimal framework. The admin pages had less interface requirements compared to the survey, allowing the structure to be simpler. Also, as a design, much of the navigational functions were moved out of the framework, and into the modules. As a result, the framework itself maintains only four factors: permission level, user attributes, graphical frame, and HTML page size.

- **Permission Level** Each administrative user was given a permission level, and was allowed to visit pages with a permission level higher than the user's level. A user with level 0, for instance, would be able to visit all pages, while a user with level 3 would only be able to see a portion of the admin pages.

- **User Attributes** In many cases, permission level does not offer a small enough granularity of control. Some administrators should be able to see all projects, but not modify the data, while other administrators should see only some of the projects but be allowed to modify the data. To allow for such flexibility, each administrative user is given a field called *attributes*. In this case, an attribute is any text value that is attached to the user. The framework itself does not make use of attributes. The modules can choose to look for any attribute they would like, and future developers are free to add as many attributes as they would like to the user. For example, an administrator can have the attribute “user=modify”. In the *PersonnelSummary* module, any user with the “user=modify” attribute will be allowed to change the information regarding researchers, while all users without this attribute can only observe the information.

- **Graphical Frame** In most cases, an admin page has a header and a footer. The header provides some graphics, the title, and some links to other pages. The footer provides some graphics and an exit link. Modules can turn off this graphical frame if needed. The function to do so is described later.
• **HTML Page Size** In most cases, the admin page has a width of 839 pixels. While this provides ample space for display on a common computer, it is too wide to be printed on a printer. To solve this problem, the width of the admin pages are actually variable. All modules are assumed to be written so that the width can be varied. Changing the value of one variable will allow the user to switch between the default 839 pixel width and the printer friendly 639 pixel width.

The framework has a simple module loading structure, based on PHP’s use of variable functions. Variable functions allow PHP to create function names at runtime and call them. By simply registering the module name in the login file, the Admin Subsystem will automatically include the module file, and start calling the functions in that file as necessary.

The interface for a module contains five functions which must all be implemented. Below is a list of the functions and what each function must do.

```plaintext
string function <module_name>_title();

This function must return the name of this module. This function is used to create the title at the top of the admin page.

void function <module_name>_navigation();

The navigation function is called to fill out a line at the top of the administration page, which is the navigational bar. Usually, this function will simply define a link or a button which allows the administrator to navigate to this page. If the navigation text needs to be changed, or if access to this page must be restricted, this method would control these issues.

boolean function <module_name>_frame();

The frame function simply returns either true or false. If true, the framework will display a border and navigation header. If false, the admin system will not display
any borders. In most cases, this function should return true. The one example where
a border is not used is with the tutorial_sample module. The tutorial_sample module
simply displays what the tutorial page looks like to an user. Therefore, the border is
not used to make the page look very much like the real tutorial page.

```c
void function <module_name>_update();
```

The update function is often the most critical part of the module. Typically, this
function would look at the request values received from the browser and perform any
necessary processing. Any form of database inserts or updates should occur within
this function.

```c
void function <module_name>_show();
```

This is the function which writes out the proper HTML and Javascript to make the
page.

## 3.3 Modules

There are many administrative modules already implemented for the system. More
are sure to be written as the project moves forward. Listed below are some of the main
modules that are already in existence. Some of the modules are for administrators
while others are for developers.

- **Program Summary** The Program Summary lists the four CMI programs in
  existence and the projects that they oversee.

- **Project Summary** The Project Summary page lists the basic information
  pertaining to a certain project as well as a list of all the project participants.
  From this page, the administrator can see the survey progress, send emails to
  participants, or access response summaries.

- **Response Summary** This summary shows the survey responses from a re-
  search participant. A research participant involved in multiple projects will
have one response summary per project. This page is essentially the same as the summary page seen by the participants at the end of the survey.

- **Personnel Directory** This page lists all researchers currently involved in any of the projects under observation. Currently, the directory is simply listed alphabetically based on their last names.

- **Personnel Summary** This page lists all information about a particular researcher. The information includes items such as name, email, phone number, position, and department. This page also lists all projects the researcher is participating in. Depending on the permission level of the administrator, the administrator can also edit these values, updating personnel records.

- **Collaboration** This page shows a matrix of all the researchers in the current project. This matrix is created from all the individual responses, and is designed to show the web of communication and collaboration between the researchers.

- **Tutorial Text** This page allows the administrators to easily change the text that accompanies the screenshots in the tutorial page.

- **Database Verifier** This module is only accessible to developers. As the system continues to run, it is possible to have inconsistencies in the data due to human error. Examples of inconsistencies are projects without any researchers, milestones for a non-existent project, and researchers who do not belong to any projects. The database verifier simply checks for any inconsistencies that exist in the database. The correction for this data is handled manually.

- **Bug Viewer** This page shows all the bugs that have been submitted by users. The page will also show the platform, browser, HTML data, and states of the user at the time of submission.

- **Printer Friendly** This module does not actually display a page. Instead, this module simply allows the administrator to switch from default width of 839 to printer friendly width of 639.
3.4 Data Issues

Most of the design issues had already been dealt with before the development of administrative and maintenance tools. The implementation of the survey subsystem had already involved many discoveries and mistakes. When I was writing the administrative modules, much of the code simply involved reusing or making modifications to the survey code. There were, however, a few issues which required considerable time and effort to solve, and I will mention them here.

3.4.1 Excel Spreadsheet Structure

The last task that was needed before the launch of the survey system was populating the database with initial data. The information that we needed were in Excel files kept by the CMI office. Once I started looking through these files, however, I realized that there were going to be some serious problems. The main problem was the lack of uniform structure to the Excel files. There were at least three different spreadsheet structures that were used, and they were all very different from each other. Furthermore, even files that were supposed to be of the same structure often contained exceptions, such as columns being switched or missing all together. Initially, I had written a script to read from a certain spreadsheet into the database, but this turned out to be insufficient. In the end, I created a process where first I would reformat the files by hand, then convert this edited excel file into a csv file and read into an intermediate database table, then finally read from the intermediate database table into the actual database table. The intermediate database table was designed to minimize the manual editing that was required. These three steps were taken for all the projects, researchers, milestones, and budget information.

3.4.2 Special Characters and Data Formatting

Even with the steps described above, there were still problems with the data itself. Some of the strings contained quotation marks or semicolons. These characters were used for special purposes in the survey system, and required filtering. More trouble-
some, however, were the milestone information. The milestones for each project was stored in one cell in the excel table. In some instances, these milestones were delimited by a period. In other instances, these milestones were delimited by a number. There were also cases where the milestones were delimited by the sigma character, a two-byte character. The delimitation was made even more difficult by the fact that periods, and numbers were also used within each milestone description. These milestones required a combination of computer parsing and manual editing to read properly into the database.

3.4.3 Checking for Human Error in the Data

The most difficult problem was detecting human error in the original Excel sheets. Mistakes were rampant, and caused many strange things to appear in the database. Some examples are misspelled last names, wrong email addresses, and failure to note personnel changes. There were no obvious programmatic way of dealing with these issues, and most were fixed manually. These errors were especially costly to fix because they often involved contacting other people to ask them for the correct information. While frustrating at time, these problems reinforced the need and motivation for an online system, where information can be observed, checked, and modified more easily.

3.5 Maintenance Issues

One major challenge of online systems is the need for constant maintenance. By maintenance, I mean both the fixes necessary for the continued execution of the system, as well as support for the current users of the system.

3.5.1 Concurrent Code Fixes

Once the system had launched, the task of fixing bugs became more constrained. I could no longer change the structure of the database tables, since users had already entered data into current formats. I also had to be more careful with my fixes, since
any new bugs introduced will directly hurt the users and result in a flood of complaint emails.

Fortunately, my project already had an infrastructure in place to handle much of this problem. We had a three-tiered development environment. The first tier is my own computer, where I could play around with the code and data without care. The second tier was the machine called Netvisdev which was essentially the staging area. On Netvisdev, any code changes I make could be tested and retested before being introduced to the public. The final tier was Netvis which was the actual server that ran the system for outside users. Netvis would have only the most stable version running.

The three-tiered system worked well. There were a few instances where I was forced to fix bugs directly to Netvis due to time constraints, but fortunately there were very few problems resulting from that. I personally feared occurrences of bugs which were platform dependent, and appeared only on Netvis or Netvisdev. Fortunately, there was only one instance of this kind of bug, which was a result from having different versions of the PHP installed in the machines.

3.5.2 Concurrent Data Fixes

Even after launch, however, there were a few instances where the database table had to be restructured. These instances posed the greatest challenge. One example of this was the comments information. Initially, the comments data did not record the project of the user entering in the comments text. If a user were to belong to two projects, however, the user would only be able to enter one set of comments for both projects. Fixing this problem required the addition of an extra column into the table.

There are many problems with editing the table after launch. First of all, the greatest care has to be used so as to not lose any of the data entered by the users. This was done by frequently backing up the data. Secondly, a user could potentially enter new data exactly at the time of database change, resulting in the loss of that new data. This was avoided by making the database update on late Saturday nights, when it seemed no one was using the system. Lastly, the original structure of the
data could potentially be plain wrong. With the comments example above, there was no automatic way to determine whether a set of comments belonged to one project or another. In this case, I manually examined the data and decided which project the comments belonged to. Fortunately, this was not much of an issue, since most users only belonged to one project, and the few that belonged to two had only filled out the survey for one project.

3.5.3 Customer Service

Immediately after the launch, there were many emails sent to the administrators. These included bug reports, suggestions, questions, constructive criticisms, and not so constructive criticisms. Simply reading these emails and responding to them would have been a time consuming and draining task. Fortunately, most of these emails were filtered before they reached me. My advisor as well as the CMI managers received and handled all the emails, and only a few bug reports ever reached me directly. Personally, I think this worked very well. At the time of launch, I already had enough bugs to keep me busy and stressed, and was not sure if I could have handled all the emails.

We did learn that a lot of time was spent inquiring the user about their platform and browser when they reported a bug. To simplify the collection of information, the bug reporting system was created, which automatically collected information about their platform and browser. We also learned that much of the emails were about confusion regarding the pages or the questions themselves. The tutorial page was written in response to these emails.

3.5.4 Bugs and Features

Another issue that came about after the launch was deciding which bugs to fix and which new features to add. Because of the risk of introducing more bugs, I was much more hesitant in making any code changes to the system. On the other hand, there was much more pressure to get certain bugs fixed and certain features added quickly,
because many users were writing in to complain. For our project, I was periodically handed down a list of bugs and wanted features. Of these, I personally decided what the priorities were. The high priority items were bugs that directly distracted the users from using the system, such as the problem with the comments database table. Medium priority items were mainly features that would greatly improve the user experience, such as the tutorial page. Any other feature request were generally left aside until later. My approach worked reasonably well. On most occasions, my priority decisions seemed to coincide with the priorities of my advisor.
Chapter 4

Conclusion

4.1 Progress Update

I signed on to the project in middle of October of 2003 and immediately started work. My progress was slow at first for various reasons, but the first prototype was more or less done by middle of December of that year. The initial prototype had all the main question types in place as well as the summary pages and the navigation system. The “Alpha” testing period, originally scheduled for late December, actually took place in the third week of January. This testing period was essentially an opportunity where potential users looked at the system briefly and give some comments about what looked good and what did not. There was very little testing of the actual functions.

The system, in its final form was done by the end of February. At this point, we received the list of projects and all research participants to be entered into the system. During this data population phase, I discovered the immediate need for administrative tools and began the development of the Administrative Subsystem. The population of the data took much longer than first anticipated, and the system did not launch until the very end of February.

Most of March was very hectic due to all the bug fixes and feature additions that required a prompt turn-around time. This was also the time when most of the admin pages were written, simply because there was so much need for them. Towards the end of March, however, the bug reports and feedback subsided, and by April most of
the wave seemed to have passed.

Currently the system is assessing 63 projects, with a combined total of 249 researchers. Of these researchers, 181 have filled out responses to the survey.

4.2 Lessons

4.2.1 Ramp-up Period

As advertised, PHP is a very simple language to learn. The syntax is straightforward, the library of functions is complete, and there is an abundance of tutorials and documentation on the Internet. There is, however, still a ramp-up period that is required to use the language effectively. Because it is a scripting language, PHP is different from Java or C++, and does not necessarily follow the design patterns common in compiled languages. For example, object-oriented programming was not used for this system, even though PHP offers Classes and Objects. This was because most of the code is simply reacting to actions by the user, and very little state is maintained, except for the data in the database. Furthermore, writing PHP code will almost always require thorough knowledge of HTML and Javascript. Learning the syntax for each of these technologies is simple, but putting them together is not always trivial. Differences in standards and implementations could lead to mysterious problems and errors.

When I began designing and writing the system, I had designed as if I was using Java or C++. This proved to be a big mistake. With PHP I have a lot less flexibility in deciding the flow of control or the behavior of the user interface, compared to writing a user application in Java. Several iterations were required before I created a PHP framework that worked around the weaknesses and made use of the strengths.

4.2.2 User Interface

User Interface is often the cause of many revisions as well as bugs. OPAS was no exception. First of all, the user interface for this system required the approval of
many people. My supervisor, the CMI office, and the Alpha testers all had explicit ideas about what the interface should do. While the multiple reviews resulted in a more complete and thorough interface, they were also a cause of confusion and delays.

The personnel pages are a good example of the interface design process. I rewrote the code for the personnel questions at least four times. First, the pages included only minimal information. The pages were then augmented to show more detailed information about each person. Next, the page was edited so that the user could modify and update the information for each person. Towards the end, however, we decided that there was too much information on the page, and the pages were stripped down to show only minimal information. In fact, the amount of information was less than what the pages showed initially.

The confusion and rewrites of the user interface is common, and difficult to avoid. What we could have done better is plan for the interface changes ahead of time. Rather than attempting to make the "final" interface each time, I could have made proof-of-concept interfaces that people could have seen and evaluated. This would have allowed for more iterations of the interface with less total time spent.

4.2.3 Standards and Implementations

One problem that I did not foresee was difficulty caused by all the different browser and platform combinations. One problem we had concerned the Internet Explorer version 6.0. With the Internet Explorer version 6.0, if a POST request timed out, the POST request is resent but with the fields empty. The result was the user would overwrite their own fields with blank data. Apart from IE6, there were also reported problems with Netscape, although these problems have not yet been reproduced or solved.

I faced problems with different browsers, even during development. For example, Internet Explorer and Mozilla based browsers had slightly different implementations of Javascript. In Internet Explorer, the HTML fields were accessible globally, while in Mozilla based browsers, the HTML fields had to be passed as an argument if they were to be used within a function.
Browser incompatibility makes the system looking very unprofessional, while resulting in frustration by the users. What we should have done is decide on browsers and platforms to support, and also create environments where testing and debugging on those platforms is possible.

4.3 Future Work

Because of the ongoing nature of the project, there is plenty of work to be done, with new feature requests coming every week. This section will describe a snapshot of the list of things that must be done for the system. Generally speaking, future work can be divided into four categories. The first category includes bug fixes, text changes and minor tweaks to the existing system. The second category is concerned mainly with the underlying structure of the system. Changes here will not be seen by the user, but will help future development and improvements. The third category, which is expected to be the largest, are features and pages needed for administrative and analytical tasks. These pages will be used by administrators as well as project researchers to gather information about the projects. The final category is major features that need to be added in anticipation of the next round of surveys.

4.3.1 Maintenance, Small Features, and Bug Fixes

While the level of activity in the section has subsided, there will always be requests to fix minor bugs, make text changes, adjust colors, etc... This is part of an ongoing improvement process to make the system better.

The main tasks left in this category are platform support and clarifying the survey pages. While the system works adequately in most windows platforms and their browsers, we have had reports of problems with Apple PowerPC computers. The problem is most likely a result of Javascript implementation differences and should not take much time. However, we currently do not have a development environment for Apple computers, and have not dealt with this issue. Some of the other necessary tasks are changing the wording of the error messages to make the more explicit, and
changing the behavior of some of the Javascript to make the pages more interactive. These are minor fixes, but should help the user experience.

4.3.2 Modularity and Cleanup

As mentioned earlier, one of the trade-offs that I made during development was the loss of modularity. In order to implement some of the last-minute interface requests, some of the modularity in the Survey Subsystem was lost. As a result, adding a new question type requires a very thorough knowledge of the system. Furthermore, there are risks that a change in one question type could affect the behavior of other question types.

Much of these interdependencies could be cleaned up. One of the goals of the project is to make the system modular enough so that other organizations could customize the framework and run their own surveys. Currently, the code requires direct code modifications to make this possible. The system should be edited so that the boundary between the Core Framework and the Survey Specific Code is better defined. The Core Framework should be reused between programs without modifications, while the Survey Specific Code should be easily modifiable, so that each program can easily create a survey system that works for them.

4.3.3 Administrative and Analytical Pages

The most pressing need right now is in Administrative and Analytical Pages. The Admin Pages currently allow administrators to view the progress of the surveys, but do not provide a thorough picture of the information collected. The next step would be to allow manipulation of this data, and allow administrators to see the data from different angles.

4.3.4 New Features

The next round of surveys are expected in about 6 months. There are several questions that need to be answered, as well as several new features that are absolutely necessary.
One feature that would be crucial is the display of past survey responses. When a user is filling out the survey next time around, the user will most likely need to refer to what they said previously. The code for this is already in place, but this feature will need to be improved and thoroughly debugged.

In the administrative side, the system will need to allow navigation not only through personnel and projects, but also through time periods. The amount of information collected will grow very quickly at this rate, and many analytical tools will be needed to sort through all this data.

Finally, the added dimension of time would no doubt introduce new challenges and questions. For example, should a user be allowed to edit previous answers?

4.4 Successes

The OPAS project had three goals. The first goal was to simplify the data collection from CMI funded projects. In this, I believe we have accomplished our goal. While there were many complaints and bug reports, the majority of the projects have already filled out their information. Furthermore, the information that we do have is more organized and more tractable. With the development of various administrative pages, the reports from the projects can be seen easily, with the desired format.

The second goal was to try to discover more about dispersed projects and their uses of technology. This goal has not yet been reached. One round of surveys is not enough data to infer trends in the projects. More surveys and more analytical tools are needed to make progress on this point.

The last goal was to develop a robust and modular framework, which can be used for ongoing surveys of CMI projects, as well as for surveys of other groups of projects. For this, I believe we are almost there. The current system is functional, and almost complete. The features necessary for online surveys are all there. What is lacking is a cleaner boundary between framework and survey specific code. One of the priorities as this project moves forward will be a more conscious approach towards separating the framework from rest of the code, allowing the system to be ported more easily to
other types of online surveys.

Finally, an underlying issue for the entire system was ease of use by the researchers. The first user responses after launch were very negative, and suggested that we were not doing a good job in this respect. As the survey period progressed, and as we made improvements to the system, the objections and complaints subsided noticeably. While, several cycles of improvements will be necessary, eventually the system should be easier and more convenient for the user than the original paper surveys.
Bibliography


Appendix A

Dear CMI Grantees,

We are pleased to unveil CMI’s new online project assessment system (OPAS). This system replaces the paper-based FP9 report, and is intended to make the reporting and assessment process simpler and more efficient. This online reporting system is designed to decrease your paperwork burden to CMI while providing valuable information about knowledge exchange in your project.

The system is also being used to collect information for a research study on knowledge exchange, which requires that you read the consent form.

We look forward to receiving your reports online,

Professor Michael J. Kelly
CU Executive Director

Professor Edward F. Crawley
MIT Executive Director

Figure A-1:index.php
Figure A-2: person_info.php
Online Project Assessment System (OPAS) Tutorial

Use the page tabs towards the bottom to move between pages on the project assessment. You can see the page name by hovering your mouse above the page number.

Use the comments field at the very bottom of the page to enter any

Figure A-3:tutorial.php
Please send any comments or suggestions to Ted Acworth (eba2@mit.edu).

**Report Overview**

**Project Assessment**

Please read each question below and select an answer of 'yes' or 'no'. Click on 'Begin' at the bottom of the page when you are ready to proceed.

**Note:** All questions are required the first time you complete the assessment

<table>
<thead>
<tr>
<th>General Information</th>
<th>Has general information about the project changed in the past 6 months?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Participation</td>
<td>Have any new CU participants, MIT participants, or Industry Partners been added to the project in the past 6 months?</td>
</tr>
<tr>
<td>Coordination and Outcomes</td>
<td>Were there examples of coordination and outcomes from the project in the past 6 months?</td>
</tr>
<tr>
<td>Budget and Work</td>
<td>Have you made changes to the cashflow, budget, funding, or statement of work in the past 6 months?</td>
</tr>
<tr>
<td>Milestones and Outputs</td>
<td>Have you made progress on key milestones, Intellectual property, public relations, or CMI goals in the past 6 months?</td>
</tr>
</tbody>
</table>

Please send any comments or suggestions to Ted Acworth (eba2@mit.edu).

Figure A-4:overview.php
If general information about the project has changed in the past 6 months, please indicate these changes below. Click on 'Next' at the bottom of the page when you are ready to proceed. (see FAQ)

<table>
<thead>
<tr>
<th>General Information</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Title:</td>
<td>projectname</td>
</tr>
<tr>
<td>Project Reference Number:</td>
<td></td>
</tr>
<tr>
<td>Reporting Period:</td>
<td></td>
</tr>
<tr>
<td>Submission Date (Today):</td>
<td>14 March 2004</td>
</tr>
<tr>
<td>Project Leader (CU):</td>
<td></td>
</tr>
<tr>
<td>Project Leader (MIT):</td>
<td></td>
</tr>
<tr>
<td>CU Project start date:</td>
<td>Sep 2003</td>
</tr>
<tr>
<td>CU Project end date:</td>
<td>Sep 2006</td>
</tr>
<tr>
<td>MIT Project start date:</td>
<td>Sep 2003</td>
</tr>
<tr>
<td>MIT Project end date:</td>
<td>Sep 2006</td>
</tr>
<tr>
<td>Total funds awarded (CU and MIT):</td>
<td>£ (- x1000)</td>
</tr>
</tbody>
</table>

Please enter comments or suggestions for this page in the textbox below.
If new Cambridge University participants have been added to the project in the past 6 months, please complete the fields under "Add CU Participant" and be sure to click the "add" button before proceeding to the next page. (see FAQ)

<table>
<thead>
<tr>
<th>Change Status</th>
<th>First Name</th>
<th>Last Name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inactivate</td>
<td>Cufirstname</td>
<td>Culastname</td>
<td><a href="mailto:cu@email.address">cu@email.address</a></td>
</tr>
</tbody>
</table>

Add CU Participant

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return to Overview

Jump to a specific page by clicking on an above page link.

Please enter comments or suggestions for this page in the textbox below
If new MIT participants have been added to the project in the past 6 months, please complete the fields under "Add MIT Participant" and be sure to click the "add" button before proceeding to the next page. (see FAQ)

<table>
<thead>
<tr>
<th>Change Status</th>
<th>First Name</th>
<th>Last Name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inactivate</td>
<td>Firstname</td>
<td>Lastname</td>
<td><a href="mailto:demo@test.test">demo@test.test</a></td>
</tr>
</tbody>
</table>

Add MIT Participant

<table>
<thead>
<tr>
<th>Add</th>
<th>First Name</th>
<th>Last Name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please enter comments or suggestions for this page in the textbox below.

Figure A-7: survey.php; MIT Participants Page
If new industry partners have been added to the project in the past 6 months, please complete the fields under "Add Industry Partner" and be sure to click the "add" button before proceeding to the next page. (see FAQ)

<table>
<thead>
<tr>
<th>Change Status</th>
<th>First Name</th>
<th>Last Name</th>
<th>Institution</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inactivate</td>
<td>Industryfir...</td>
<td>Industrylas...</td>
<td>IndustryPar...</td>
<td><a href="mailto:ind@email.a">ind@email.a</a>...</td>
</tr>
</tbody>
</table>

Add Industry Partner

<table>
<thead>
<tr>
<th>add</th>
<th>First Name</th>
<th>Last Name</th>
<th>Institution</th>
<th>Email</th>
</tr>
</thead>
</table>

Please enter comments or suggestions for this page in the textbox below.
If you worked directly with other participants in the past 6 months, please select how often you worked with them before proceeding to the next page. (see FAQ)

<table>
<thead>
<tr>
<th>Participant</th>
<th>Institution</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cufirstname Culast...</td>
<td>Cambridge University</td>
<td>frequently</td>
</tr>
<tr>
<td>Industryfirstname ...</td>
<td>IndustryPartner</td>
<td>infrequently</td>
</tr>
</tbody>
</table>

Figure A-9: survey.php; Participant Collaboration Page
If you organized work differently for the project in the past 6 months, please check all that apply and give a specific example for each box you check before proceeding to the next page. (see FAQ)

Think about the different ways you organized work over the past 6 months...

<table>
<thead>
<tr>
<th>Please Check All That Apply</th>
<th>Please Give Specific Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>□ Subgroups worked on different tasks/studies</td>
<td></td>
</tr>
<tr>
<td>□ Implemented project manager role in project</td>
<td></td>
</tr>
<tr>
<td>□ Faculty directly supervised tasks/studies</td>
<td></td>
</tr>
<tr>
<td>□ Post-doc supervised tasks/studies</td>
<td></td>
</tr>
<tr>
<td>□ Grad student supervised tasks/studies</td>
<td></td>
</tr>
<tr>
<td>□ Common lab space</td>
<td></td>
</tr>
<tr>
<td>□ Common lab equipment</td>
<td></td>
</tr>
<tr>
<td>□ Common web site</td>
<td></td>
</tr>
<tr>
<td>□ Common dataset</td>
<td></td>
</tr>
<tr>
<td>□ Held conference</td>
<td></td>
</tr>
<tr>
<td>□ Held workshop</td>
<td></td>
</tr>
<tr>
<td>□ Held seminar</td>
<td></td>
</tr>
<tr>
<td>□ Invited outside speaker</td>
<td></td>
</tr>
<tr>
<td>□ Hosted visitor at site</td>
<td></td>
</tr>
<tr>
<td>□ Offered multidisciplinary course</td>
<td></td>
</tr>
<tr>
<td>□ Other [ ]</td>
<td></td>
</tr>
</tbody>
</table>

Please enter comments or suggestions for this page in the textbox below.

Figure A-10: survey.php; Organization of Work Page
Think about the communication and travel you had over the past 6 months...

<table>
<thead>
<tr>
<th>Please Check All That Apply</th>
<th>Please Give Specific Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monthly face-to-face meetings with all participants</td>
<td></td>
</tr>
<tr>
<td>Monthly face-to-face meetings with PIs only</td>
<td></td>
</tr>
<tr>
<td>Monthly face-to-face meetings with students only</td>
<td></td>
</tr>
<tr>
<td>Email at least once a month</td>
<td></td>
</tr>
<tr>
<td>Telephone at least once a month</td>
<td></td>
</tr>
<tr>
<td>Conference call at least once a month</td>
<td></td>
</tr>
<tr>
<td>Video Conferencing at least once a month</td>
<td></td>
</tr>
<tr>
<td>Instant messenger at least once a month</td>
<td></td>
</tr>
<tr>
<td>Online forum discussion at least once a month</td>
<td></td>
</tr>
<tr>
<td>Worked on project during a conference or workshop</td>
<td></td>
</tr>
<tr>
<td>Worked on project during sabbatical leave</td>
<td></td>
</tr>
<tr>
<td>Drove car to another site to work on project</td>
<td></td>
</tr>
<tr>
<td>Flew on airplane to another site to work on project</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td></td>
</tr>
</tbody>
</table>

Figure A-11: survey.php; Communication and Travel Page
Think about the student training and project outreach over the past 6 months...

Please Check All That Apply

- Undergrad/grad student finished thesis or dissertation
- Undergrad/grad student/post-doc got academic job
- Undergrad/grad student/post-doc got industry job
- Undergrad went to grad school
- Formed partnership with industry
- Formed community relationships through research
- Formed collaborations with different researchers
- Other

Please Give Specific Example(s)
If there were examples of new ideas or tools from the project in the past 6 months, please check all that apply and give a specific example for each box you check before proceeding to the next page. (see FAQ)

Please Check All That Apply

- Started new field or area of research
- Developed new models or approaches in field
- Came up with new grants or spinoff projects
- Submitted patent application
- Presented at conference or workshop
- Published articles, books, or proceedings
- Recognized with award for contribution to field
- Developed new methodologies
- Created new software
- Created new hardware
- Generated new datasets
- Improved infrastructure at site
- Other

Please Give Specific Example(s)
CMI is obligated to provide estimated monthly cash flows to the Treasury. In order to enable us to construct a reasonably accurate report, please describe any significant monthly factors, over £50,000, that might affect your spending in the next 12 months, such as purchasing equipment, summer salaries, etc.
Give below your best estimate of the expected quarterly cash-flow (to the nearest £'1000) for your project for the next 12 months. If there are any budgetary changes that you have identified, please incorporate these into this projection.

<table>
<thead>
<tr>
<th>Budget Line</th>
<th>First Quarter</th>
<th>Second Quarter</th>
<th>Third Quarter</th>
<th>Forth Quarter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total CU</td>
<td>£0</td>
<td>£0</td>
<td>£0</td>
<td>£0</td>
</tr>
<tr>
<td>Total MIT</td>
<td>£0</td>
<td>£0</td>
<td>£0</td>
<td>£0</td>
</tr>
</tbody>
</table>

Nov 2003 - Jan 2004  
Feb 2004 - Apr 2004  
May 2004 - Jul 2004
If additional sources of funding have been identified in the past 6 months, please respond in the area below before proceeding to the next page. (see FAQ)

Do you see any additional sources of support, both financial and non-financial, on the horizon: from government, private-sector, or philanthropic sources? Can CMI help you get them?
If you made significant changes to the statement of work in the past 6 months, please respond in the area below before proceeding to the next page. (see FAQ)

Please advise if you have made any alterations to the project design or statement of work since final application or last report (please detail any information regarding the project that was not included in your final application for CMI).
If you made progress on any of the project milestones in the past 6 months, please use the pull-down menu to the left of each milestone to update the completion progress. To add a new milestone please complete the fields under "Add Milestone" and be sure to click the "add" button before proceeding to the next page. (see FAQ)

Figure A-18: survey.php; Key Milestones Page
Has any intellectual property (patents, licences, journal articles, copyright) been created as a result of the project in the past 6 months? If so, please describe the IP below and be sure your principle investigator or project manager is aware of the IP.

Figure A-19:survey.php; Intellectual Property Page
Please give details of any accomplishments or outputs that may enable us to publicize your project more widely. For example, is there any novel knowledge exchange process that could be shared publicly?
If there are any significant events happening in the next 6 months, please respond in the area below before proceeding to the next page. (see FAQ)

Please give dates and descriptions of any expected activities in the future (such as conferences at which your research will be discussed, high-profile meetings, the production of prototypes, etc).
Please describe how this project is contributing to the goals of CMI and to UK competitiveness (Call for Proposals).
Appendix B

Implementation Details of the Survey System

This section will illustrate the details of the survey system. First I will describe all the tables in the database, explaining the structure and purpose for each table. I will then describe the groups of code and what they do.

B.1 Database Structure

Like many online applications, the backbone of the system is the database. The database structure defines much of what can and can not be done with the system. The database structure and the queries also dominate much of the computation and performance issues. The following is some of the main database tables used in the Survey System. Tables B.1, B.2, and B.3 show the complete list of database tables used in the system. Below is more indepth description of some of the more important tables.

- **User Table** This table is responsible for storing information specific to a person. Each researcher in the system has exactly one entry in this table. This table will contain information such as name, phone number, address, title, etc...
<table>
<thead>
<tr>
<th><strong>Table Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>project</td>
<td>Contains project data. One entry per project.</td>
</tr>
<tr>
<td>user</td>
<td>Contains researcher data. One entry per person in the system.</td>
</tr>
<tr>
<td>people</td>
<td>Contains project specific data on the researchers. One person will have an entry for each participating project.</td>
</tr>
<tr>
<td>questions</td>
<td>Contains the actual questions that can be asked in the survey.</td>
</tr>
<tr>
<td>question_pages</td>
<td>Contains the pages that can be in the survey. Each page can contain multiple questions, but typically only contains one.</td>
</tr>
<tr>
<td>question_sections</td>
<td>Contains the groups of pages that make up a survey. Each section contains one or more pages. One page can be in multiple sections. User permissions are set at a section basis. A researcher</td>
</tr>
<tr>
<td>results</td>
<td>Contains the survey results from each person. will have one entry for every project they are participating in.</td>
</tr>
<tr>
<td>session_data</td>
<td>Caches user progress so that the user can pick up where they left off when they relogin.</td>
</tr>
<tr>
<td>comments</td>
<td>Contains any extra information, messages, written by the users.</td>
</tr>
<tr>
<td>bug_reports</td>
<td>Contains any bug information submitted by the users.</td>
</tr>
<tr>
<td>access</td>
<td>Records platform and browser used by the users.</td>
</tr>
<tr>
<td>strings</td>
<td>Contains strings displayed in various places in the system.</td>
</tr>
<tr>
<td>email_messages</td>
<td>Contains email texts which can be sent by the system.</td>
</tr>
<tr>
<td>tutorial_strings</td>
<td>Contains strings displayed in the tutorial page.</td>
</tr>
<tr>
<td>faq</td>
<td>Contains FAQ question and answer.</td>
</tr>
</tbody>
</table>

Table B.1: Tables in the Database Used by the Framework

<table>
<thead>
<tr>
<th><strong>Table Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>people_activity</td>
<td>Contains any modifications made by users to the people table.</td>
</tr>
<tr>
<td>milestone</td>
<td>Contains information regarding milestones in a project.</td>
</tr>
<tr>
<td>milestone_activity</td>
<td>Contains modifications made by users to the milestone table.</td>
</tr>
<tr>
<td>budget</td>
<td>Contains information regarding the cash-flow of a project.</td>
</tr>
<tr>
<td>budget_activity</td>
<td>Contains modifications made by users to the milestone table.</td>
</tr>
</tbody>
</table>

Table B.2: Tables in the Database Used for the Questions

<table>
<thead>
<tr>
<th><strong>Table Name</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>people_table</td>
<td>Staging area for importing user data from Excel files.</td>
</tr>
<tr>
<td>project_supplement</td>
<td>Staging area for importing milestone and budget data.</td>
</tr>
<tr>
<td>project_table</td>
<td>Staging area for importing project data from Excel files.</td>
</tr>
<tr>
<td>administrators_table</td>
<td>Maintains account information for administrators.</td>
</tr>
</tbody>
</table>

Table B.3: Other Tables in the Database
• **Project Table** This table stores all basic information pertaining to a project. The fields in this table are project name, statement of work, reference number, funding, and project schedule.

• **People Table** While the name might be misleading, this table describes which researcher belongs in which project. This table also contains information about a person’s role and access level on a per-project basis. The combination of the People Table and the User Table allows the possibility of someone being a Principal Investigator of one project while being a participant in another.

• **People Activity Table** When a Principal Investigator edits personnel information for their project, they do not edit the information directly in the People Table or the User Table. Instead, that entry is copied onto this People Activity Table, and the changes are made in here. In this way, two Principal Investigators could make different updates and decisions about the same person. The activity table also allows easy review of what people have changed and what people have not changed.

• **Questions Table** This table lists all the questions that are asked in the survey. In theory, an addition of a new question requires a new entry in this table and some modification in the Question Section Table described below.

• **Question Sections Table** One section is a group of questions that must be answered by a researcher. A section has one permission level, meaning if a person must answer one question in a section, the person must also answer all other questions in the section. The sections are also listed in the overview page, giving the researchers a broader view of the survey as a whole.

• **Results Table** Finally, the results table contain user responses for the survey. While some survey questions, such as personnel and milestones edit other specialized tables, text questions and checkbox questions all save their responses in an entry in this Results Table.
B.2 Code

The PHP code in the system can be roughly divided into framework code and survey specific code. As described earlier, the framework code allows the user to log in, select a project, verify background information, and proceed to the survey. The survey specific code handles the actual questions that are asked, and this code is expected to be different from survey to survey.

As a general rule, the HTML generating code and data handling code is separate. Consequently, a lot of the functionalities have two files. Milestones, for example, has milestone.php which generates the HTML code for the milestone question. Milestones also has milestone_submit.php, which actually handles the submission of data from the user. In contrast, the administration system contains both the HTML generation and data handling in the same file. I have come to think that the second approach is cleaner, due to simpler dependencies, and less code files, and will probably move the survey system to that model soon. For now, however, many functionalities will have two separate files.

- **index.php** This file is essentially the entry point for all users. Apart from having the user type the username and password, this page also initializes all global variables. Furthermore, if a user tries to navigate directly to any other page before logging in, that user is automatically forwarded to this page.

- **login.php** This file simply checks the username and password to log in. If the two fields match, the user's session variables are set. If the fields do not match, the user is sent back to the index page.

- **person_info.php** Whenever the user logs in, the system checks if all the background information for the user has been filled in. If not, the user is first sent to this page. The person_info.php page essentially creates a form where the user can enter or edit all of their background information. This information is then sent to person_info_submit.php where the fields are checked for correctness and saved.
<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index.php</td>
<td>Displays the index page.</td>
</tr>
<tr>
<td>login.php</td>
<td>Checks the user and password.</td>
</tr>
<tr>
<td>person_info.php</td>
<td>Displays the background page.</td>
</tr>
<tr>
<td>person_info_submit.php</td>
<td>Handles the data submission from person_info.php.</td>
</tr>
<tr>
<td>select_project.php</td>
<td>Display the project selection page. Only displayed if the user has multiple projects.</td>
</tr>
<tr>
<td>start_survey.php</td>
<td>Restores user state from last time they were logged in.</td>
</tr>
<tr>
<td>overview.php</td>
<td>Display the survey section selection page.</td>
</tr>
<tr>
<td>overview_submit.php</td>
<td>Handles data submitted in overview.php.</td>
</tr>
<tr>
<td>tutorial.php</td>
<td>Display the tutorial page.</td>
</tr>
<tr>
<td>submit.php</td>
<td>Handles user input for survey questions.</td>
</tr>
<tr>
<td>survey.php</td>
<td>Displays a survey page.</td>
</tr>
<tr>
<td>printer_survey.php</td>
<td>Displays a printable survey page.</td>
</tr>
<tr>
<td>summary.php</td>
<td>Displays the summary page, showing all user responses.</td>
</tr>
<tr>
<td>printer_summary.php</td>
<td>Displays a printable summary page.</td>
</tr>
<tr>
<td>verify.php</td>
<td>Handles user verification at the summary page.</td>
</tr>
<tr>
<td>reminder.php</td>
<td>Displays a reminder window, which will email the user with username and password.</td>
</tr>
<tr>
<td>bug_report.php</td>
<td>Displays the bug report page, allowing the user to submit bugs.</td>
</tr>
</tbody>
</table>

Table B.4: PHP Files Used by the System
• **overview.php** This file displays the groups of questions, called sections, the user will be asked. The user can select which sections they would like to work on. For first time users, however, they are required to select all the sections regardless. The selections are sent to overview.submit.php where the survey is initialized. After the survey has been created and various survey related variables are set, the user is forwarded to survey.php where the actual survey will begin.

• **survey.php** This is the main survey page. Depending on the current page, this file will show different content to the user. This page is essentially the framework for the survey, with each question type being a module which fits within the framework.