DPAS: The Dynamic Project Assessment System

by

Birendro M. Roy

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

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2005 [June 2005]

Author ....
Department of Electrical Engineering and Computer Science
May 19, 2005

Certified by ....
Jonathon Cummings
Assistant Professor
Thesis Supervisor

Accepted by .. ..
Arthur C. Smith
Chairman, Department Committee on Graduate Students

DPAS: The Dynamic Project Assessment System

by

Birendro M. Roy

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, I designed and implemented the second iteration of a web-based system that administers survey questions to users and aggregates responses. Several survey system components were reimplemented according to an updated software architecture, and several new system components were developed. Chief among these is a web-based administration console that provides a unified access method for several frequently used functions. Individual console functions are built around an object-oriented framework that will serve as a basis for the next system design iteration.

Thesis Supervisor: Jonathon Cummings
Title: Assistant Professor

Acknowledgments

First and foremost, I would like to thank my project and thesis supervisor, Professor Jonathon Cummings, for the opportunity to work on Project Assessment. He kept me on task over the course of the project and was always willing to provide encouragement and objective feedback.

I also would like to thank my academic advisor, Professor Patrick H. Winston, for a few well-timed words of wisdom. Without his patient advice, I would not have made it through my years at MIT.

Furthermore, I would like to acknowledge the many outstanding faculty at the Institute, especially in the department of computer science. This project would not have been as successful as it was if I had not received such an outstanding education.

I would like to thank my coworkers at the Initiative for Distributed Innovation for their support, collaboration and positive feedback throughout the project.

Finally, I would like to thank my mother, father and sister for their love and for always believing in me.

Contents

1 Introduction **13**

1.1 Overview of Project Assessment 13

1.2 Objectives of the Dynamic Project Assessment System 14

1.3 Evolution of the Online Project Assessment System 15

1.3.1 The Original 15

1.3.2 Supporting Multiple Organizations 17

1.3.3 Multi-part Surveys 20

1.4 Improvements made by the Dynamic Project Assessment System . . . 21

1.4.1 Offsite Installations 21

1.4.2 Administrative Tools 22

1.5 Technical Requirements 23

1.5.1 Standards Compliance 23

1.5.2 Backwards Compatibility 23

1.5.3 Anticipating Future Needs 24

2 Redesigning the Survey System **27**

2.1 Guiding Principles 27

2.2 Model-View-Control 28

2.2.1 Introduction to Model-View-Control 29

2.2.2 Application to Survey Page Flow 31

2.2.3 Application to Survey Configuration 32

2.3 Rethinking Modularity 34

2.3.1 Web Page Design 34

2.3.2 OPAS Modules 35

3 Survey Construction **37**

3.1 Overview 37

3.2 Motivation 37

3.3 The Survey Administration Console 38

3.3.1 Survey Questions 38

3.3.2 Survey Sections 41

3.3.3 Special Question Types 46

3.3.4 Survey Periods 47

3.3.5 Survey Settings 49

3.4 Extending the Console 51

4 Conclusion **55**

4.1 Challenges Overcome 55

4.2 Lessons Learned 57

4.2.1 Collaboration 57

4.2.2 Time Management 58

4.2.3 Development vs. Deployment 58

4.3 Future Directions 58

4.3.1 Web Page Design 59

4.3.2 Architectural Improvements 59

4.3.3 Open Source Release Work 60

4.4 Final Words 61

A Source Code Examples **63**

List of Figures

1-1 Page flow in OPAS 16

1-2 OPAS page flow with multiple organizations 18

1-3 Page flow in the Dynamic Project Assessment System 21

2-1 Model-View-Control component interactions 29

2-2 Raw access to the data tier vs. unified access via a data object 30

2-3 Hierarchy of View components 31

2-4 Control flow in the Survey-Submit loop 31

2-5 Sample rows from the `settings` table. 33

3-1 Survey Administration Console login page 39

3-2 Survey Administration Console main menu 39

3-3 Survey Administration Console question list page 40

3-4 "Edit Question" page for questions not using the `question_config` table 42

3-5 "Edit Question" page for questions that use the `question_config` table 43

3-6 Survey Administration Console "Edit Survey" page 44

3-7 Survey Administration Console "Edit Section" page 45

3-8 Sample rows from the `general_info_questions` table. 46

3-9 Selecting a Survey Period 48

3-10 The table editing GUI in action 50

Listings

3.1 Javascript functions used by Table Editor. 52

A.1 *Settings* class source code. 63

A.2 *Strings* table source code. 66

A.3 *surveyadmin.AdminPage* base class. 68

A.4 *surveyadmin.TableEdit* extension of *surveyadmin.AdminPage*. 72

A.5 Administrative page framework: `admin_page.php`. 78

A.6 Administrative page framework: `admin_page_submit.php`. 80

Chapter 1

Introduction

With each passing year, new advances in information technology bring people around the world closer together. Businesses have embraced this shrinking of the globe with open arms; multinational corporations provide the ultimate in economies of scale. Scientific and academic communities have also benefited from more powerful communication tools. As a result, more organizations are bringing together distributed workgroups, or research and project teams that span multiple geographic locations.[6]

Management scientists have long studied effective organizational structures within a company or division at a single location, but research on distributed workgroups is still a relatively young field. One of the primary challenges of working in this space is the difficulty of collecting raw data. Paper surveys were once the tool of choice for researchers in this field, but the disadvantages of paper are apparent even when collecting data on traditional workgroups; these disadvantages are multiplied when workgroups are not constrained to a single location. The Project Assessment web-based survey system was developed to address the needs of management scientists studying distributed workgroups.

1.1 Overview of Project Assessment

The original survey system was developed by Fumiaki Shiraishi for the purpose of distributing surveys to members of the Cambridge-MIT Institute (CMI). This system, dubbed the Online Project Assessment System (OPAS), allowed Professor Jonathon Cummings at the Sloan School of Management to design a survey, distribute it to project members at CMI, aggregate responses, and export the resulting data for further analysis. [10]

Surveys built using OPAS were intended to function in a manner similar to paper surveys, but with considerably less hassle. Each page of the web survey corresponds to a single survey question. Users are able to navigate through the survey pages in any order, and revisit their answers as many times as necessary. Responses are saved immediately upon submitting a survey page or when the user logs out. Users can fill out a partial response, then log in later to complete the survey.

Survey administrators manage the users, projects and survey response data for CMI using an administrative console written for OPAS. This console provides basic administrative functions such as: adding and deleting users to and from projects, sending reminder e-mail to users, and viewing response rates.

The survey web pages are written in PHP, one of the most popular web scripting languages. All OPAS data are stored in a MySQL database; these data are made available to the survey system web pages via PHP's built-in library of MySQL functions. [3]

1.2 Objectives of the Dynamic Project Assessment System

Although OPAS in its original form satisfied the needs of the Cambridge-MIT Institute, it required several extensions and patchwork fixes as more organizations with diverse objectives were added to the mix. Over time, old functionality became obsolete and new features were added in a haphazard fashion. The primary objective of the Dynamic Project Assessment System is to provide all of the functionality of the previous survey system in a reengineered package. In addition, the Dynamic Project Assessment System aims to achieve the following objectives:

1. **Better support for web standards.** The web pages generated by DPAS are compliant with the W3C HTML 4.01 and CSS 2.1 standards.

2. **A more flexible survey question structure.** Survey questions use a more intuitive and expressive method of configuration.

3. **A more user-friendly survey management interface.** DPAS provides a completely redesigned administration console that allows uniform administration of all organizations participating in the survey.

4. **More support for secondary administrators.** There are more tools that allow the offloading of administrative tasks onto managers or HR personnel at the surveyed organizations.

5. **Automated installation and initialization.** Many DPAS web pages and data components automatically initialize themselves to a default state when no configuration information is found. This behavior provides a useful example survey for new developers and administrators to build upon, while avoiding mysterious error messages when the survey configuration is changed or deleted.

1.3 Evolution of the Online Project Assessment System

The Project Assessment survey system has evolved considerably from its first incarnation into the system that is used today. I will first describe several key properties of the original Project Assessment web application, then I will identify how these properties have changed, sometimes breaking the assumptions of the original design.

1.3.1 The Original

The original OPAS supported a single, relatively fixed survey; once the content and appearance of a survey was laid out, it was expected to require minimal maintenance.

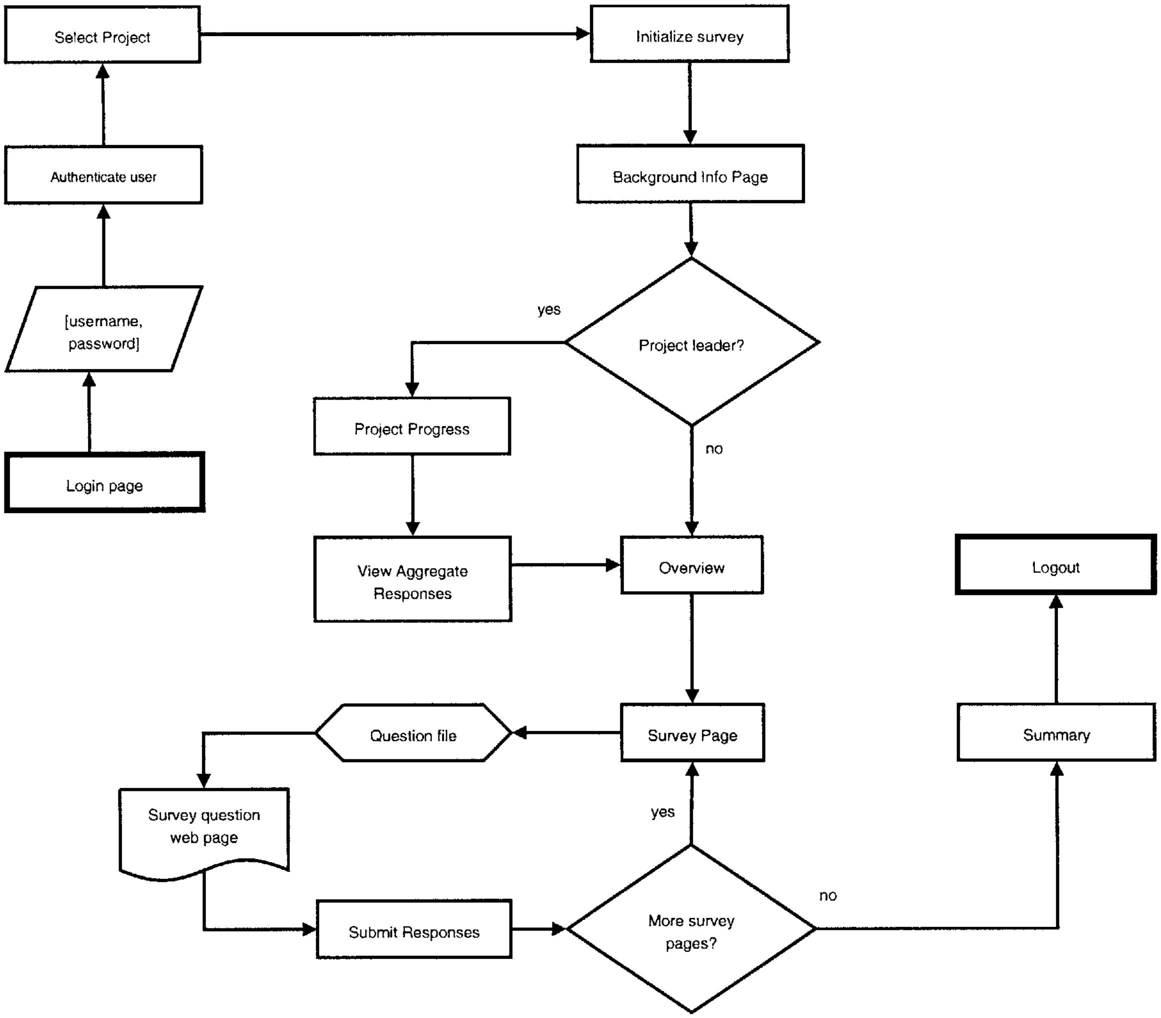


Figure 1-1: Page flow in OPAS

It was therefore acceptable for the initial setup of a survey to require a significant amount of time and specialized knowledge of the survey system internals, since it was not expected that this task would be performed frequently. Though the CMI survey was expected to solicit responses for several survey periods, the number and types of questions were not expected to change from one survey period to the next. Ideally the survey would not change at all over time, so that responses from later periods could be compared sensibly with data from prior periods to identify statistical trends and interrelationships.

Although multiple surveys were not explicitly supported, the original Project Assessment system allowed for multiple "views" of a survey based on a survey respondent's permission level. This was most useful when the survey designers wanted to show certain classes of users only a subset of the full survey. For example, principal investigators or project leaders would see the entire survey, project members would see a slightly smaller subset, and minor contributors or project advisors would only be asked a handful of questions. Because all survey responses (regardless of view) were written to the same results table, multiple views as implemented in the original survey system only made sense if there was a significant overlap between the questions asked in the various views.

Originally, the Project Assessment software was designed to support only a single organization: the Cambridge-MIT Institute. As with the survey creation process, the steps to set up the software for a new organization were not streamlined, since it was not expected to happen frequently. Modifying the look and feel of an organization required access to the filesystem of the web server so that new images and stylesheets could be uploaded. Some per-organization settings were loaded from a PHP configuration file; editing these settings also required filesystem access.

OPAS provided an administrative console written specifically to manage the CMI survey. This console allowed CMI administrators to view project member information, send reminder e-mails, add or delete users, and analyze response rates. New functionality was added on a regular basis by writing new modules according to the OPAS administrative console template system. The various functionalities provided by the console were implemented by administrative modules, which are written around an OPAS-specific template system that was created to speed the development of new survey components.

1.3.2 Supporting Multiple Organizations

The initial success of OPAS at collecting data from CMI project members allowed Professor Cummings to recruit other organizations to participate as part of Project Assessment. Distributing surveys to multiple organizations required a website that could present different surveys with a distinct look and feel depending on which organization a user belonged to. There were also privacy concerns; a participating

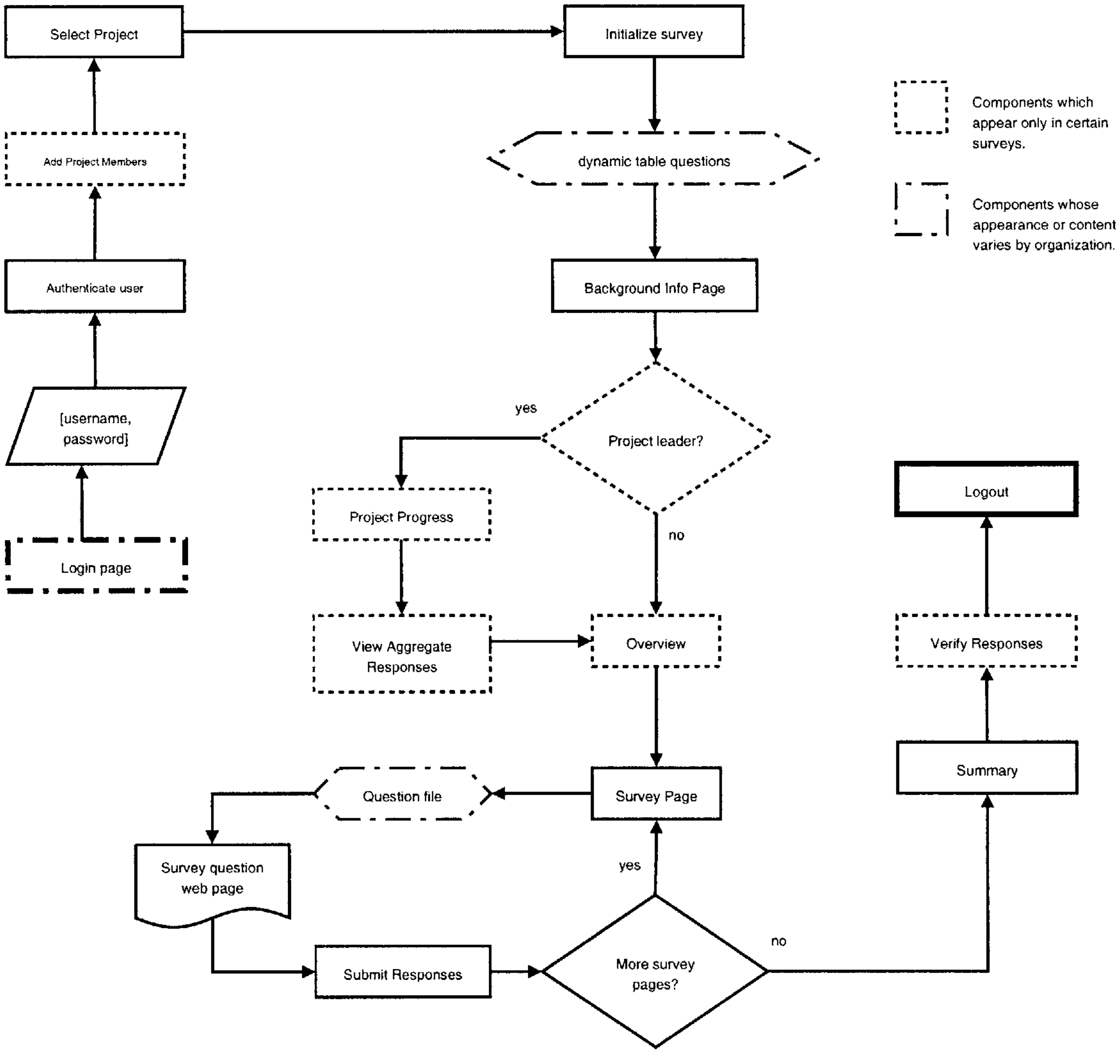


Figure 1-2: OPAS page flow with multiple organizations

organization might not want to be identifiable to Project Assessment users from other organizations.

Each organization received its own subfolder on the web server. The name of the subfolder usually reflected an "organization ID" derived from the name of the organization. For example, the CMI survey is accessible via the URL: `http://www.projectassessment.org/cmi/`. The survey for "Company A" might be located in subfolder `http://projectassessment.org/ca/`. In the original OPAS implementation, this organization-specific subfolder contained the following items:

- `images/` Subfolder containing images displayed on survey pages, including a header image or company logo, left and right arrows, and a form submit button.

- `stylesheet.css` CSS stylesheet used to format the survey pages according to the desires of the particular organization.

- `settings.php` PHP file containing various configuration options used by the survey system.

- `index.php` The login page seen by a user who navigates to an organization-specific URL.

Survey data for each organization, including the survey questions, user data and survey responses, were stored in a MySQL database named with the "organization ID." Certain tables were common to every database. The most important tables were:

- `user`: Contains information about each user, including username, password and full name.

- `projects`: Contains information about each project, including project name, project leader and start and end dates.

- `people`: Maps users to projects. A user may be on more than one project; in this case he or she will have multiple records in this table.

- `periods`: Lists the start and end dates of survey response periods.

- `question_sections`: Defines sections of questions and specifies the permission level required to view each section.

- `questions`: Defines individual survey questions. Each row corresponds to a single question, and contains all information required to specify the behavior of the question.

- `results`: Contains survey responses. Each row is identified by a user ID number and a project ID number.

- `strings`: Defines values for survey text that changes based on the organization.

Survey settings are no longer loaded from a PHP file in DPAS; instead, they are now loaded from a new `settings` database table. The migration of survey settings from PHP to a database backend was driven by a desire to provide a unified access method to all survey-related configuration options.

In the original OPAS, new organizations were not expected to be added often, so there was no automated "New Organization" wizard provided. Typically, a new organization was added by performing the following steps:

1. Create a subfolder for the new organization, copying files from an existing organization.

2. Edit the stylesheet to reflect the desired look and feel of the new organization.

3. Edit the settings file to enable or disable survey behavior according to the desires of the surveying organization.

4. Copy database tables from an existing organization to a new database.

5. Edit the survey sections and survey questions to include the desired items.

6. Populate the `user`, `projects`, and `people` tables with data provided by the new organization.

DPAS includes an organization management console that exposes the ability to create new organizations and reconfigure existing ones via a convenient web interface.

1.3.3 Multi-part Surveys

Two-part survey support has only recently been added to the Project Assessment system, at the request of one of the participating organizations. For this particular organization, a second survey page sequence displays if the response for the first part meets certain conditions. Because multi-part surveys were not part of the problem domain for the original system, they were added only upon request and implemented in this limited fashion. As the Project Assessment system gains in popularity, it is

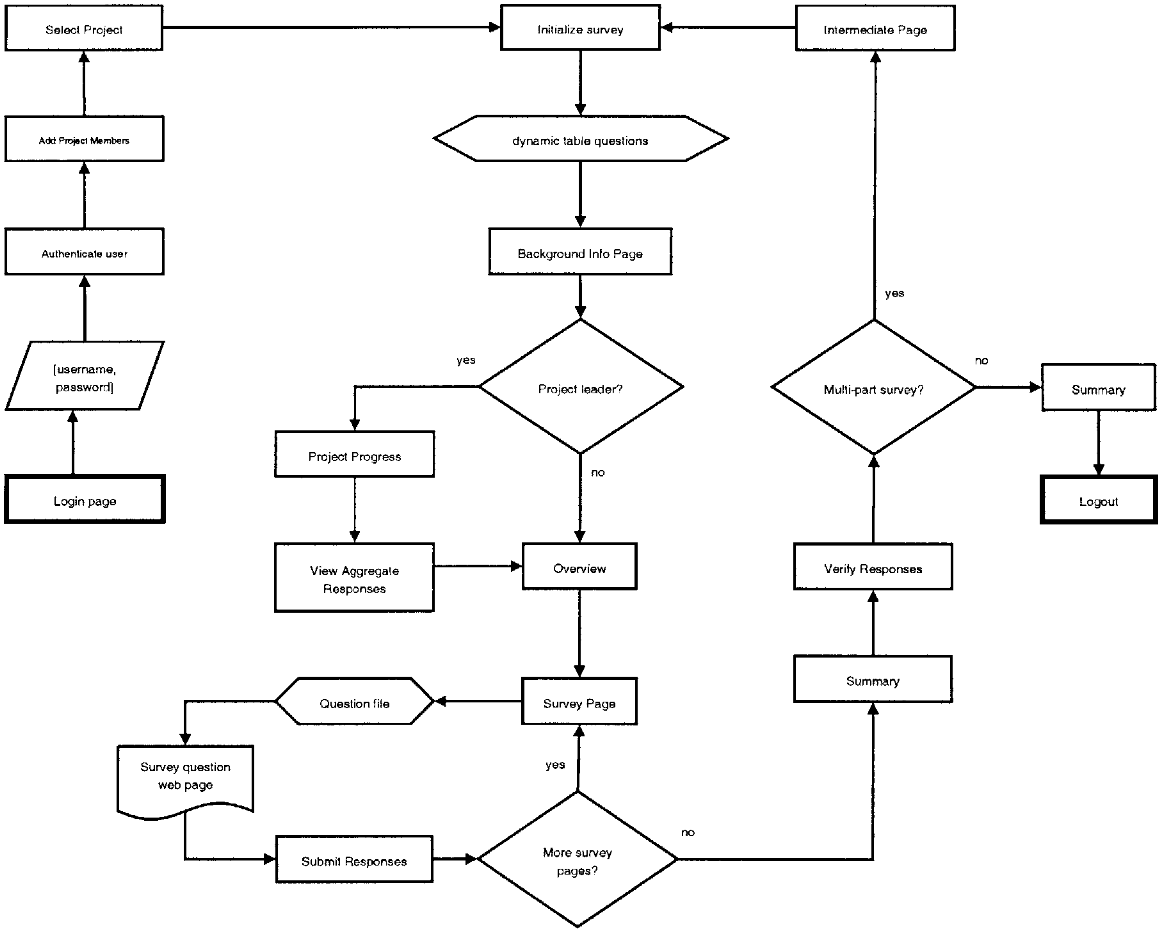


Figure 1-3: Page flow in the Dynamic Project Assessment System

more and more likely that there will be other organizations that require support for multi-part surveys.

1.4 Improvements made by the Dynamic Project Assessment System

1.4.1 Offsite Installations

Some participating institutions might decide to run a subset of the Project Assessment software on internal servers for security or availability reasons. Because this arrangement requires more expense in terms of manpower and equipment, we expect that in most cases organizations will want us to host and administer the survey.

In those situations where the Project Assessment system must be deployed offsite, however, we must have the ability to do so quickly and easily.

Directly participating organizations are only one possible offsite scenario. Ever since the origin of Project Assessment, Professor Cummings has had the intent to release the survey system source code and database specifications as an open source package. Before making the project available to the public, however, the system required a step-by-step installation process and ancillary documentation. In the past, there was no fixed, documented process; we simply archived the PHP source code and exported the SQL database by hand whenever the need arose. DPAS provides three major improvements upon the old system:

1. A standard `README` file listing system requirements and installation steps.

2. A survey packaging tool that administrators can use to create exported standalone versions of surveys they have authored using DPAS.

3. An automatic repackaging tool that generates a current snapshot of PHP source code and exported SQL data from a MANIFEST file on demand.

1.4.2 Administrative Tools

Managing the multiple organizations participating in Project Assessment had become a challenge in and of itself by the time development on DPAS began. Each new organization comes with its own administrative overhead in the form of survey configuration, management of users and projects, bug tracking and implementation of feature requests. At the time, Project Assessment personnel performed most of these functions by browsing and modifying MySQL database tables directly using phpMyAdmin. This strategy required very little effort to implement; once installed, phpMyAdmin could be used to browse or edit any database table. The primary disadvantage of this approach was that it required the ability to generate (or memorize) several complex SQL statements to perform conceptually simple tasks, like setting a user's status to "inactive" or changing the current survey period.

DPAS provides an administrative console that implements these features and more. The Survey Administration Console was built in order to provide a single interface for all of the functions an administrator might need to perform within the scope of a single organization. The SAC is driven by an object-oriented framework that is described in more detail in section 3.4. Since Project Assessment is continually evolving, this framework is designed in a way that encourages the development of new modules that inherit functionality from existing components.

1.5 Technical Requirements

1.5.1 Standards Compliance

As more and more organizations join Project Assessment, the user base of the system will grow considerably. Experience has shown that the technology with which survey respondents access the internet varies considerably. The best way to ensure a consistent experience for all users of the survey system is to adhere to established web standards. In this case, the most important documents are the World Wide Web Consortium's specifications for HTML 4, CSS 1 and CSS 2.1. [12, 14, 13] Although different browsers will occasionally render even fully standards-compliant web pages differently, the W3C specifications provide an excellent target for web development.

1.5.2 Backwards Compatibility

As of May 2005, a total of nine organizations have committed to using the Project Assessment system to distribute electronic surveys to their members. Although these organizations publish surveys on their own schedules, at least one survey is active at any given time. For this reason, active development of the survey system must maintain continuity for existing organizations as much as possible. Unfortunately this makes it difficult to enact broad changes or rewrites to the survey code base. Adding new features rarely causes a problem; as long as they are optional, older surveys can simply ignore them. Backwards compatibility causes the most difficulty when there is a desire to rewrite the underlying code in order to realign the software architecture of the system. Often this limitation required us to phase in an architectural change gradually as manpower allowed. For example, when the new object-oriented table access methods were developed, the old procedural access methods were left in place until all dependent code could be updated.

The data architecture of the survey system is similarly resistant to change. Surveys that have already completed one response cycle are especially tricky, since any changes to the database structure carry the risk of accidental damage to data from prior survey responses. Even if data are not damaged directly during an update, old survey responses may become less useful if the assumptions under which survey data are gathered have changed significantly.

1.5.3 Anticipating Future Needs

While inventing solutions to existing challenges, we must be sure not to build rigid systems which will themselves become roadblocks to progress later on. One way to avoid this pitfall is to use design patterns appropriate to the situation that have stood the test of time. Another way is to plan carefully and have in mind specific extensibility requirements from the start.

The Project Assessment surveys already use a modular framework for survey questions and administrative modules, but that only guarantees a consistent user experience when viewing questions or administrative pages. There are many other pages, such as the Overview or Select Project pages, that are not able to be represented within the modular question framework. Survey pages in DPAS have been refactored to decouple the PHP and HTML code to the greatest extent possible; this reorganization should aid future efforts to bring the entire survey page flow into a modular page hierarchy.

DPAS has also attempted to avoid assuming that an organization will only have one active survey. It may be that in the future an organization will want to conduct simultaneous surveys that target different departments, for example. New features such as the multi-organization administration console and the new style of question configuration have been designed to be adaptable if and when Project Assessment must support multiple concurrent active surveys.

Chapter 2

Redesigning the Survey System

The design and implementation phases of the software development cycle are necessarily more constrained on the second iteration than the first.

Section one delineates the guiding principles of the redesign effort.

Section two describes the Model-View-Control design pattern and how it applies to our survey application.

Section three describes an improvement upon the existing module system for survey questions and administrator modules.

2.1 Guiding Principles

Less is more. In the world of engineering in general, it is desirable for the solution to a problem to be of minimal complexity. A simple, streamlined solution is preferable to a more complicated one for many reasons: it is easier to understand and maintain, less likely to contain bugs and usually more efficient. Software engineers achieve minimally complex solutions in a variety of ways, but one primary method is to reuse components wherever possible. When redesigning the survey system, I attempted to identify and eliminate instances of duplicated functionality.

Modular solutions allow more complex behaviors. When adding a feature to a system that supports multiple instances, the developer has a choice as to whether to add the feature to a specific instance or to the system as a whole. In the case of our survey system, this translates to the choice between adding a feature to a particular organization's survey, or to the base survey system. In general, the best approach is to add the feature to the underlying system, and allow it to be turned on or off for individual instances. There is a combinatorial motivation for this approach: consider three features A, B and C. Rather than writing eight code paths that implement all non-trivial combinations of the three features, it makes more sense to implement the features as separate modules and call a module if and only if its corresponding switch is "turned on." [7, pp. 163-165] Top-down design techniques can help a developer achieve this kind of modularity. [4, pp. 143-144, 271-272]

Design for the target audience. Users, administrators and developers all have different needs. An interface designed for a developer can assume more knowledge of the underlying system than one designed for an end user. When building new survey system components and updating existing ones, care was taken to consider the skills and limitations of an average member of the target audience. In several cases, Javascript was used to augment existing question types or OPAS pages to make them more intuitively navigable. [8]

2.2 Model-View-Control

The Model-View-Control (MVC) design pattern was originally created as a logical framework for developing graphical user interfaces alongside applications written in the Smalltalk language. In the MVC paradigm, application data, logic and user interface components are explicitly defined and distinguished. The clear division of functionality made it easier for Smalltalk developers to build applications with complex flows of control, multiple views and several data types. [5, 11, 9]

Early dynamic web applications were typically developed as monolithic collections of documents written in the developer's choice of scripting language or template system. As web applications grew in scope and complexity, they became harder

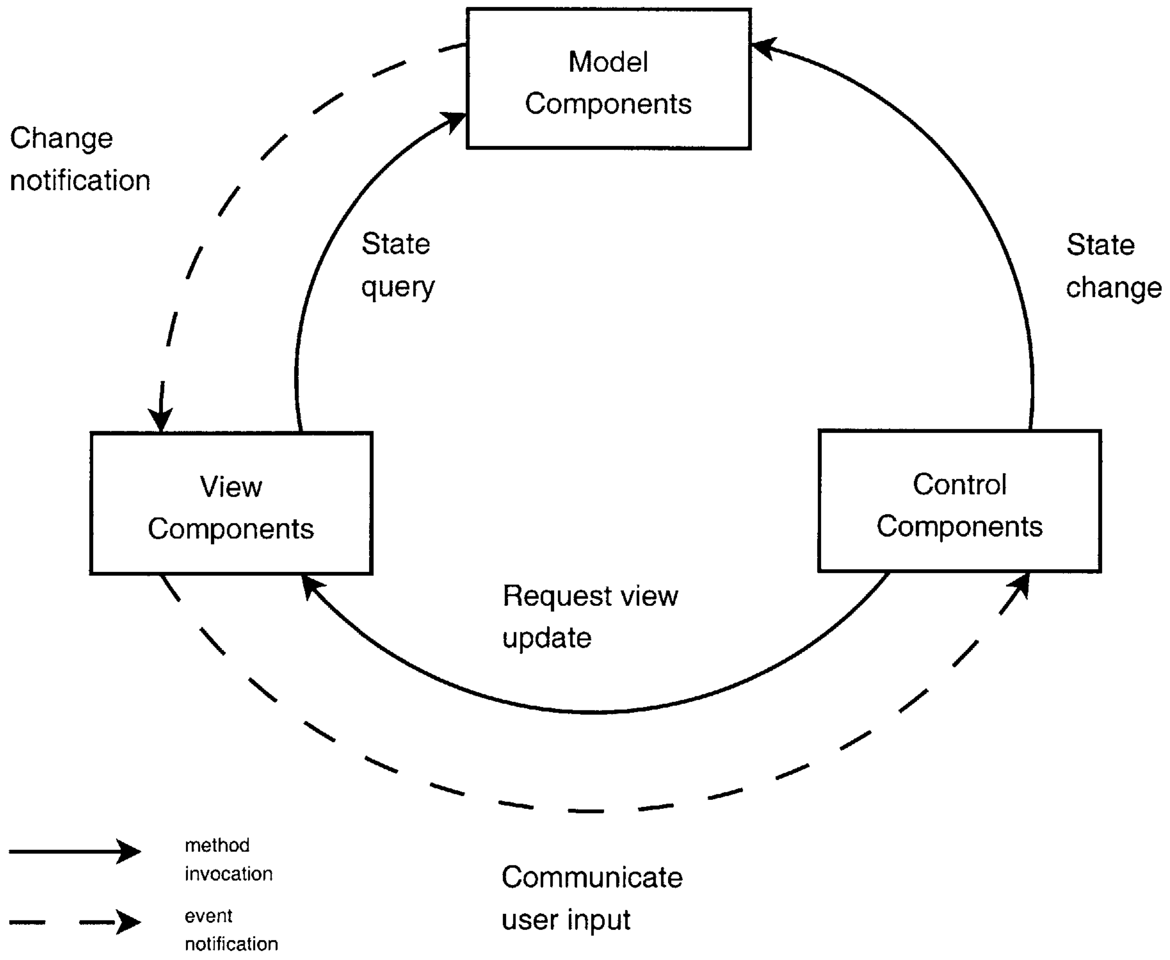


Figure 2-1: Model-View-Control component interactions

to design, implement and maintain. A few insightful web developers realized that providing a modular, open source development framework could simplify all three tasks. In May 2000 they started the Apache Struts Project [2], which was the first MVC framework for Java/JSP web applications. Web applications developed using such a framework became known as "Model 2" applications, by way of contrast with the more tightly coupled applications written prior to the advent of such frameworks, which are referred to as "Model 1."

2.2.1 Introduction to Model-View-Control

In an MVC architecture, application data or state is maintained by Model components. In an object-oriented language, these components often take the form of classes

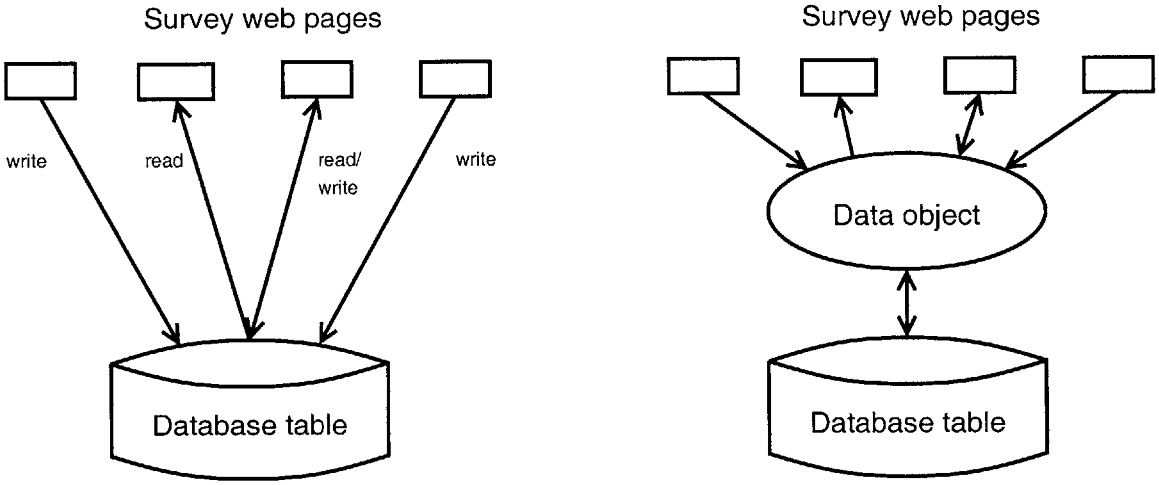


Figure 2-2: Raw access to the data tier vs. unified access via a data object

that wrap data representations like streams or arrays. If we apply this concept to tiered web applications, it makes sense to wrap database queries, tables and records in object classes. Access to and operations on data then occur over an interface that abstracts away details of how the information is stored. If this interface is properly defined, adding support for an additional data source should require at most one new class. Furthermore, changes in the underlying representation of a data source affect only the wrapper class.

View components are responsible for interacting directly with the user. This requires displaying data provided by Model components as well as informing Control components of user inputs. In a web environment, the single most complex component of the user's interaction with the application is the web browser. The content developer does not have any direct control over the user's choice of browser, however, so traditionally the set of View components is restricted to the application components generating dynamic HTML, as well as CSS stylesheets and any other UI components such as Java applets or Javascript code blocks. In the Project Assessment code base, all dynamic HTML is generated by PHP pages.

Control components tie the Model and View components together. In complex web applications with sets of pages that can be navigated in an arbitrary order, Control components handle transitions between web pages. Form submission and related

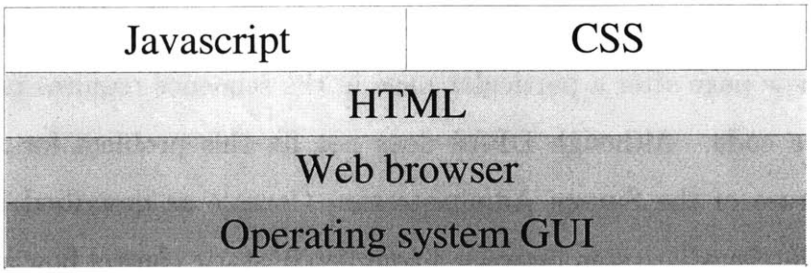


Figure 2-3: Hierarchy of View components

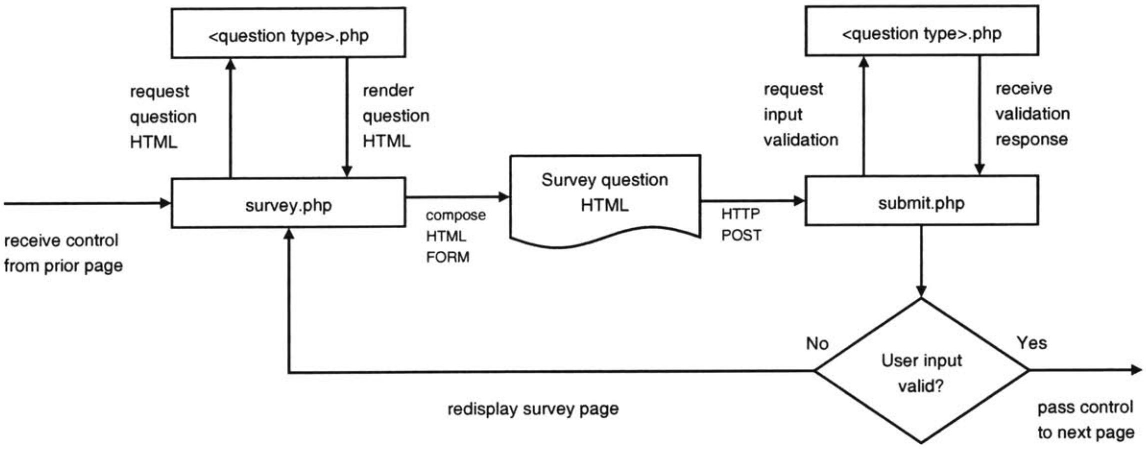


Figure 2-4: Control flow in the Survey-Submit loop

updates to Model components might also be the responsibility of Control components. In MVC frameworks like Struts, Control components describe how incoming HTTP requests are processed and directed to View components.

2.2.2 Application to Survey Page Flow

In a web application built according to the principles of Model-View-Control, each incoming request is handled by an Action object. These objects typically contain parameters that determine how requests are handled and are usually defined in an XML configuration file by the application developer. Action definitions also specify where control is forwarded once the request processing code returns a success or failure. In this way the flow of control through the web application is determined *externally* by an explicit configuration, rather than within application components.

The flow of control through the Project Assessment survey system in both of its incarnations (OPAS and DPAS) has been determined by static links between pages. Inserting a new page after a particular page in the sequence requires modifying that page's source code. Although DPAS does not fix this problem for survey pages, the architecture of the Survey Administration Console as described in section 3.4 provides an intermediate step between a completely static control flow strategy and a fully configurable one. Future work on DPAS as described in section 4.3.2 will include improvements to the representation of control flow in the survey system.

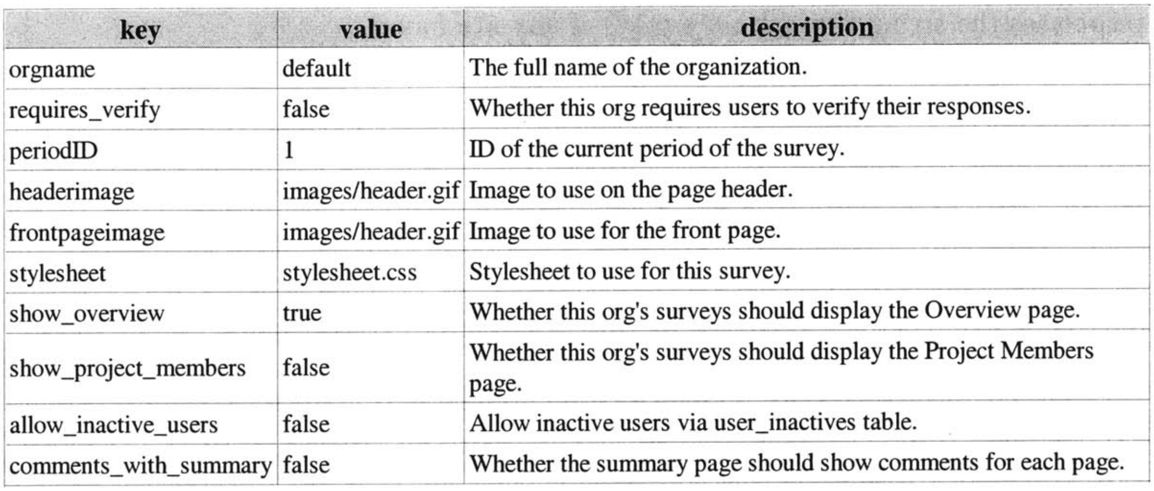
2.2.3 Application to Survey Configuration

Settings Table

One of the primary goals for the redesigned survey system was to migrate the per-organization survey settings from a PHP configuration file to a database table. This would allow consistent access to all aspects of the survey; modifying survey settings uses the same process as modifying survey pages.

In line with the Model 2 design philosophy, the new `settings` table is wrapped with a *Settings* object. When a new object is instantiated, it checks the organization's `settings` table against a built-in list of default *(key, value)* pairs. If any keys are missing from the table, it creates a new entry based on the default value for each key. This automatic repair functionality helps to avoid errors caused by missing survey setting keys. Since survey settings are not expected to change once a survey has launched and users are interacting with the system, the *Settings* object need only be instantiated once per user session. This avoids unnecessary accesses to the database and may improve the scalability of the survey system.

There are three access methods for survey settings, depending on whether the client is requesting a string, a boolean value or a URL. Although all values are stored in the database as strings, the boolean and URL access methods provide some preprocessing before returning values from the database. `Settings.getFlag($key)` returns true if and only if the value stored in the database is string-equal to `"true"`. `Settings.getURL($key)` returns the value from the database, prepended with the



key	value	description
orgname	default	The full name of the organization.
requires_verify	false	Whether this org requires users to verify their responses.
periodID	1	ID of the current period of the survey.
headerimage	images/header.gif	Image to use on the page header.
frontpageimage	images/header.gif	Image to use for the front page.
stylesheet	stylesheet.css	Stylesheet to use for this survey.
show_overview	true	Whether this org's surveys should display the Overview page.
show_project_members	false	Whether this org's surveys should display the Project Members page.
allow_inactive_users	false	Allow inactive users via user_inactives table.
comments_with_summary	false	Whether the summary page should show comments for each page.

Figure 2-5: Sample rows from the `settings` table.

current organization ID. These specialized access methods allow the client of the *Settings* object to look up a setting without having to interact with the `settings` table via SQL statements, check for errors, and convert the resulting value.

The *Settings* object provides a mutator that allows the caller to provide the value for a particular setting without worrying about whether that setting already exists in the table. `Settings.set($key, $value)` will automatically check for the existence of an entry for `$key`, using an SQL INSERT if the value does not exist and an SQL UPDATE if it does.

Strings Table

Due to the varying requirements of the organizations participating in the Project Assessment surveys, many strings displayed on survey pages must be configurable on a per-organization basis. To provide for this functionality, there exists a `strings` table for each organization that stores strings by key. In order to simplify access to this table, it was wrapped in a *Strings* class similar in functionality to the *Settings* class above. The *Strings* class, however, provides only a single access method and no mutator. Modifications to the `strings` table are expected to be performed manually, so no programmatic mutator is required. The single access method `Strings.get($key)` checks the value associated with `$key` for variable names (indicated with $) and interpolates the string through an `eval()` if any are found.

2.3 Rethinking Modularity

The Model-View-Control pattern and its applicability to the design of web applications has demonstrated that modularity is an important tool when developing complex systems. The following sections explore the application of modular design principles to existing elements of the survey system.

2.3.1 Web Page Design

HTML and CSS are two basic standards that underpin all web sites, whether static or dynamic. Although HTML was used to represent both form and content during the early days of the World Wide Web, the World Wide Web Consortium (W3C), an international standards organization, has since developed the Cascading Style Sheet (CSS) specification as a richer, more flexible method of describing the aesthetic attributes of a web page. Currently, the W3C recommends that to the fullest extent possible, HTML be used to encapsulate a page's content, and CSS be used to describe its appearance to the user.

Properly factoring a web site into mutually independent HTML and CSS documents is challenging, but has immediate benefits. As part of the Project Assessment survey system, survey pages have a "Printer Friendly" view that disables form input elements and converts the page into a portrait orientation for easy printing. In the ideal case, switching between the "Printer View" and "Normal View" would require nothing more than switching the CSS stylesheet used to style the page. The original survey web site used HTML attributes as the primary method used to format survey pages. As an intermediate step, nearly all HTML-based formatting has been replaced by inline CSS via the `STYLE` attribute.

2.3.2 OPAS Modules

OPAS survey questions were initially implemented using a module system whereby the PHP filename of a survey question was used to locate question-specific functions within a global namespace. Questions supplied four primary methods: `<filename>_show()`, `<filename>_summary()`, `<filename>_submit_tag()` and `<filename>_handle()`. OPAS administrator modules written for the CMI administration console used a similar set of methods: `<filename>_title()`, `<filename>_navigation()`, `<filename>_update()` and `<filename>_show()`. These module systems made it easy to write new question types and administrator modules, but suffered a major pitfall: if a survey developer wanted two pages that displayed slightly different behavior, there were two options. First, the original question type could be copied to a new file, the function names changed, and the source code slightly modified. This has the benefit of keeping both resulting question types relatively simple, but at the expense of considerable code duplication and double the maintenance. Another option would be to add a switch to the original question type, allowing for the selection of one behavior or the other. This solution avoids code duplication but requires more effort to implement.

Those familiar with object-oriented languages will know that this problem has already been solved by object hierarchies. In an object-oriented system, the developer would add a new question type by extending a base question type and writing new code only for behavior that differs from the base. Each page in the Survey Administration Console is implemented by a PHP subclass of *surveyadmin.AdminPage*, and in the future survey pages will also be adapted to an object-oriented architecture.

Chapter 3

Survey Construction

3.1 Overview

This chapter describes a new web-based Survey Administration Console (SAC). The goal of the SAC is to drastically reduce the time Project Assessment personnel spend designing and configuring a survey.

Section two describes the motivaton for designing and implementing the SAC.

Section three describes the capabilities of the console itself.

The final section describes the software architecture of the console, and explains how additional functionality can be implemented by building on existing components.

3.2 Motivation

Ever since OPAS was developed, constructing a survey required in-depth knowledge of the survey source code. Since each survey question type was implemented differently, the format of an entry in the `questions` table depended heavily upon which question type was being configured. Some question types required more configuration than others, so the finite fields of the `questions` table were forced to don different meanings when used with different questions. Adding a question to an existing survey by inserting a `questions` table row required either reading the source code or finding an example to build upon.

Building survey sections was slightly easier, since the names of fields in the `question_sections` table closely reflect their actual meanings. Unfortunately the process was still not very user-friendly, since the questions that belong to a particular section are specified by question number. Survey administrators editing the `question_sections` table had to flip back and forth between it and the `questions` table in order to resolve question number cross-references.

Although once surveys are finalized and launched they typically require only minimal changes, experience with prior surveying organizations has shown empirically that surveys undergo several revisions as we prepare them for the customer. If Project Assessment ever added more than two new organizations at a time, the necessary cycles of revision combined with omnipresent maintenance work would quickly become too much for one person to handle. In short, we desperately needed a tool that made the process of setting up a survey easy and intuitive.

3.3 The Survey Administration Console

Survey administrators begin using the SAC by logging in with a username, password, and organization ID. The login page then searches for a record with the specified username and password in the `administrators` table in the database named by the organization ID. Different organizations can therefore have different administrators; although for now all survey edits are performed by Project Assessment staff, this functionality will be useful if any part of the survey design process is offloaded onto external administrators.

Upon a successful login, the user is directed to the Main Menu. All SAC functionality is available from this menu.

3.3.1 Survey Questions

Selecting "List and Configure Survey Questions" from the Main Menu will bring the user to a page listing questions by question number, question title and question type. This view is very similar to examining the `questions` database table directly, but

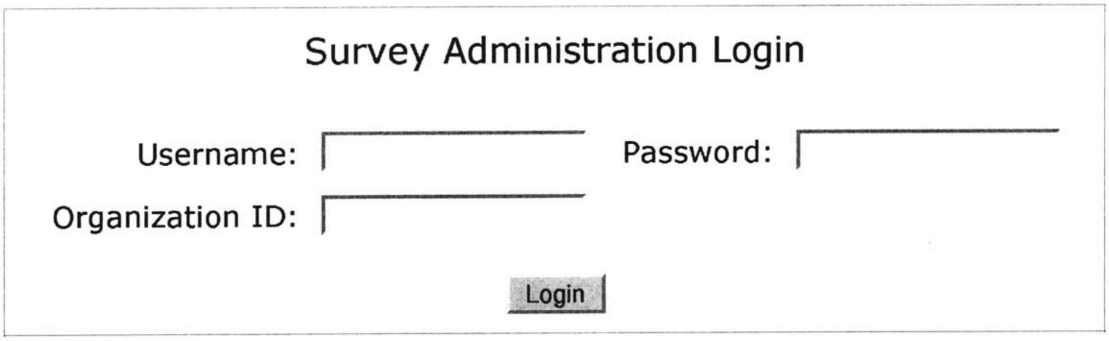


Figure 3-1: Survey Administration Console login page

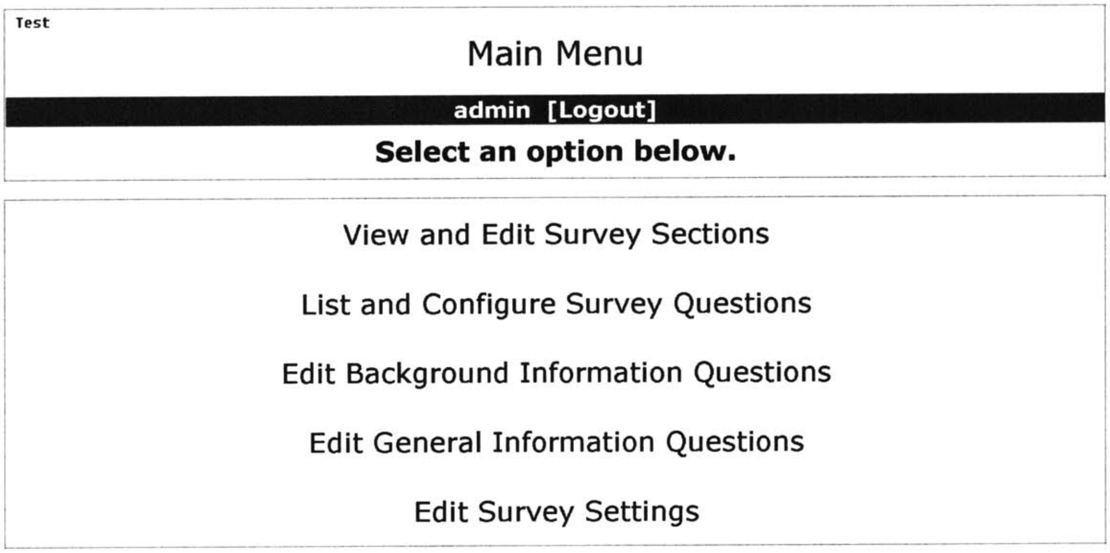
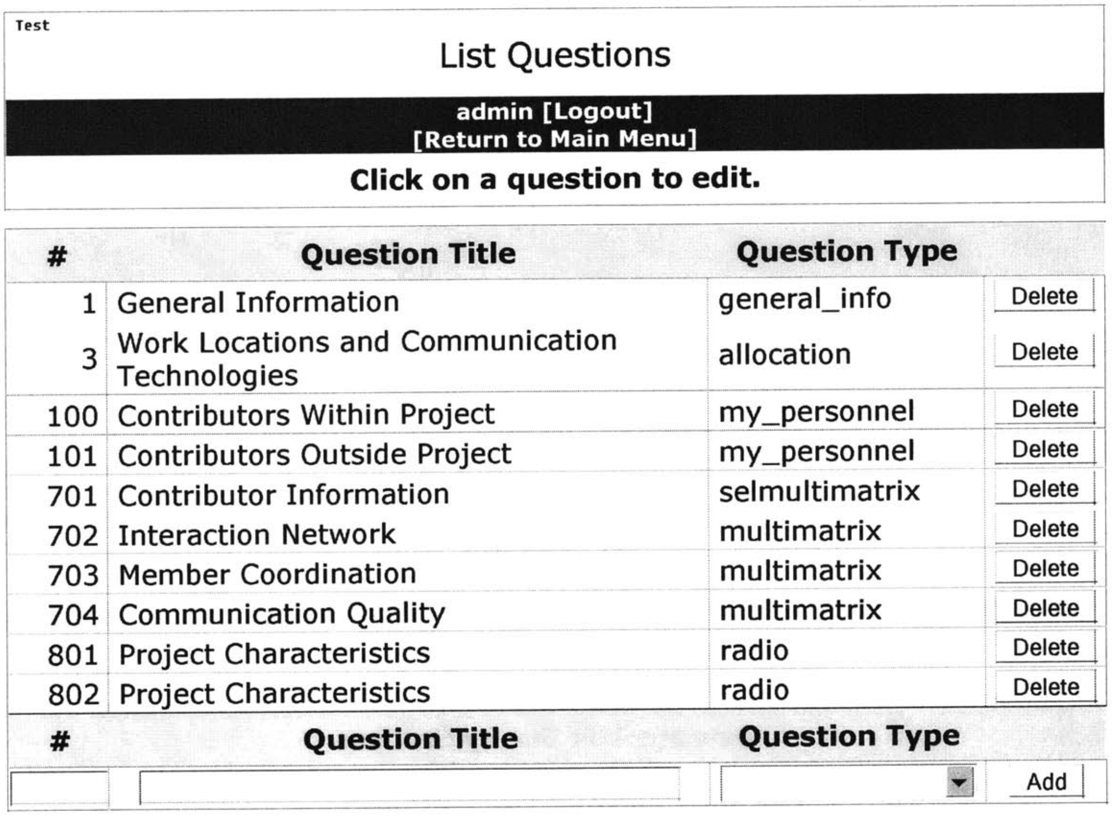

Test
Main Menu
admin [Logout]
Select an option below.
View and Edit Survey Sections
List and Configure Survey Questions
Edit Background Information Questions
Edit General Information Questions
Edit Survey Settings

Figure 3-2: Survey Administration Console main menu



Test

List Questions

admin [Logout]
[Return to Main Menu]

Click on a question to edit.

#	Question Title	Question Type	
1	General Information	general_info	Delete
3	Work Locations and Communication Technologies	allocation	Delete
100	Contributors Within Project	my_personnel	Delete
101	Contributors Outside Project	my_personnel	Delete
701	Contributor Information	selmultimatrix	Delete
702	Interaction Network	multimatrix	Delete
703	Member Coordination	multimatrix	Delete
704	Communication Quality	multimatrix	Delete
801	Project Characteristics	radio	Delete
802	Project Characteristics	radio	Delete
#	**Question Title**	**Question Type**	
			Add

Figure 3-3: Survey Administration Console question list page

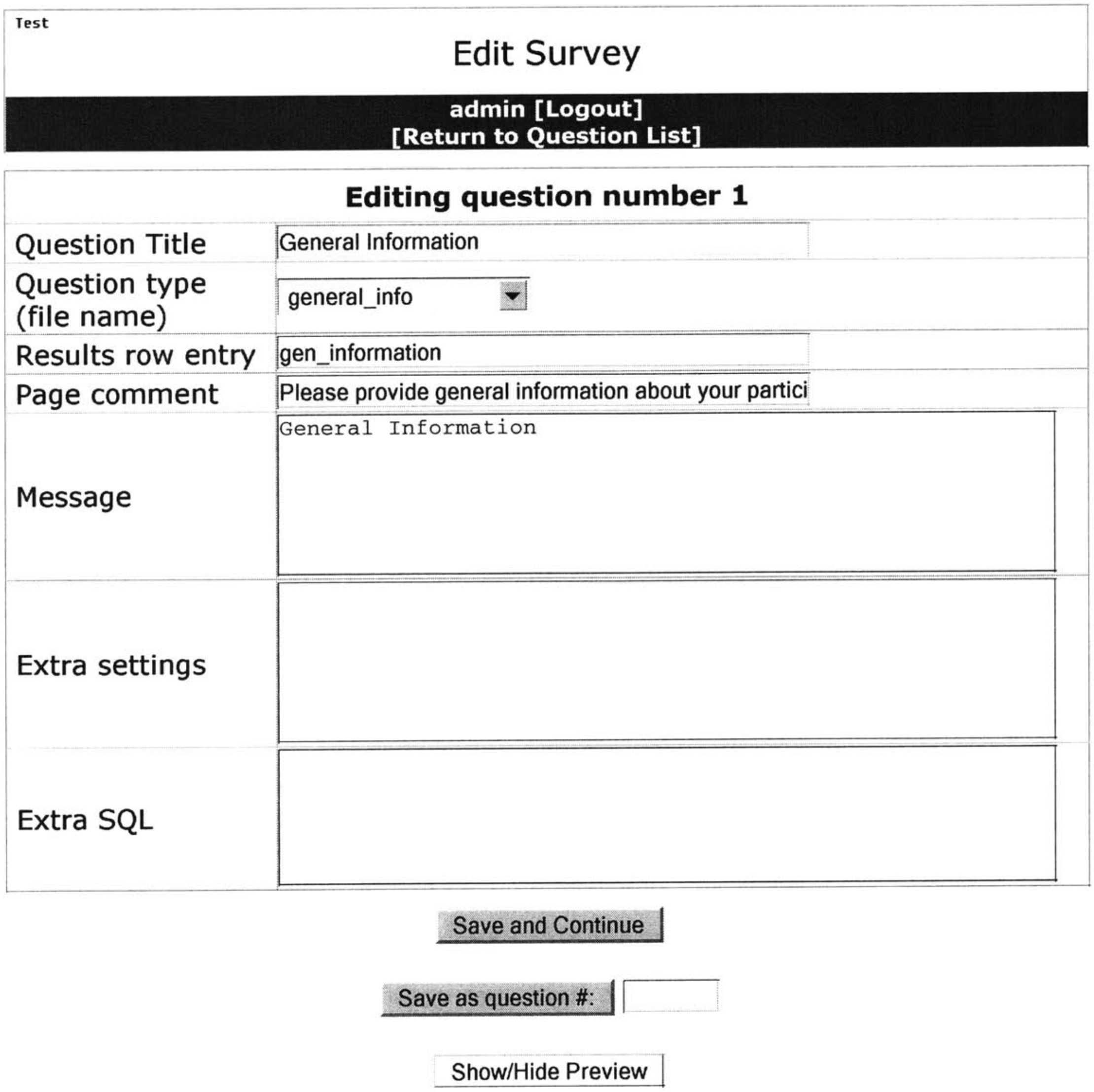
with a less cluttered layout and the addition of user-friendly features. From this page one can easily add a question by filling out the input fields at the bottom of the page, or delete a question by clicking on the corresponding "Delete" button. Clicking on a question title selects a question for editing and forwards the user to the "Edit Question" page. Adding a question will create a new row in the `questions` table and then forward the user to the "Edit Question" page.

The appearance of the "Edit Question" page differs depending on whether the question being edited supports the new type of question configuration or not. If it does not, the page shows editable fields corresponding to the `Label`, `Message`, `extra`, `question_title`, `file_name`, `sql_extra` and `comment` fields from the `questions` table. Editing these fields is analogous to modifying the database table directly, but the user is given the additional options of saving the modified values as a new question number, or previewing the question page as it will appear to people taking the survey.

If the question type supports the new style of question configuration, the only directly editable fields from the `questions` table are `Label`, `question_title`, `file_name` and `comment`. In lieu of the remaining fields, a table of configuration items and their values is presented to the user. Configuration items are loaded by selecting all rows from the `question_config` table corresponding to the current question number. The display format of each configuration item depends on its declared type in the `question_config` table. Boolean values are represented as a pair of True/False radio buttons. Text values are rendered as a text area. Lists display as a vertical column of text boxes. Table values appear as two-dimensional tables. By allowing for an arbitrary number of configuration items per question, displaying values in intuitively understandable formats and showing the user a preview of the question page, we simplify considerably the task of configuring a question according to the desired specifications.

3.3.2 Survey Sections

Selecting "View and Edit Survey Sections" from the Main Menu brings the user to a page listing survey sections. This page loads the fields `sectionID`, `permission`,

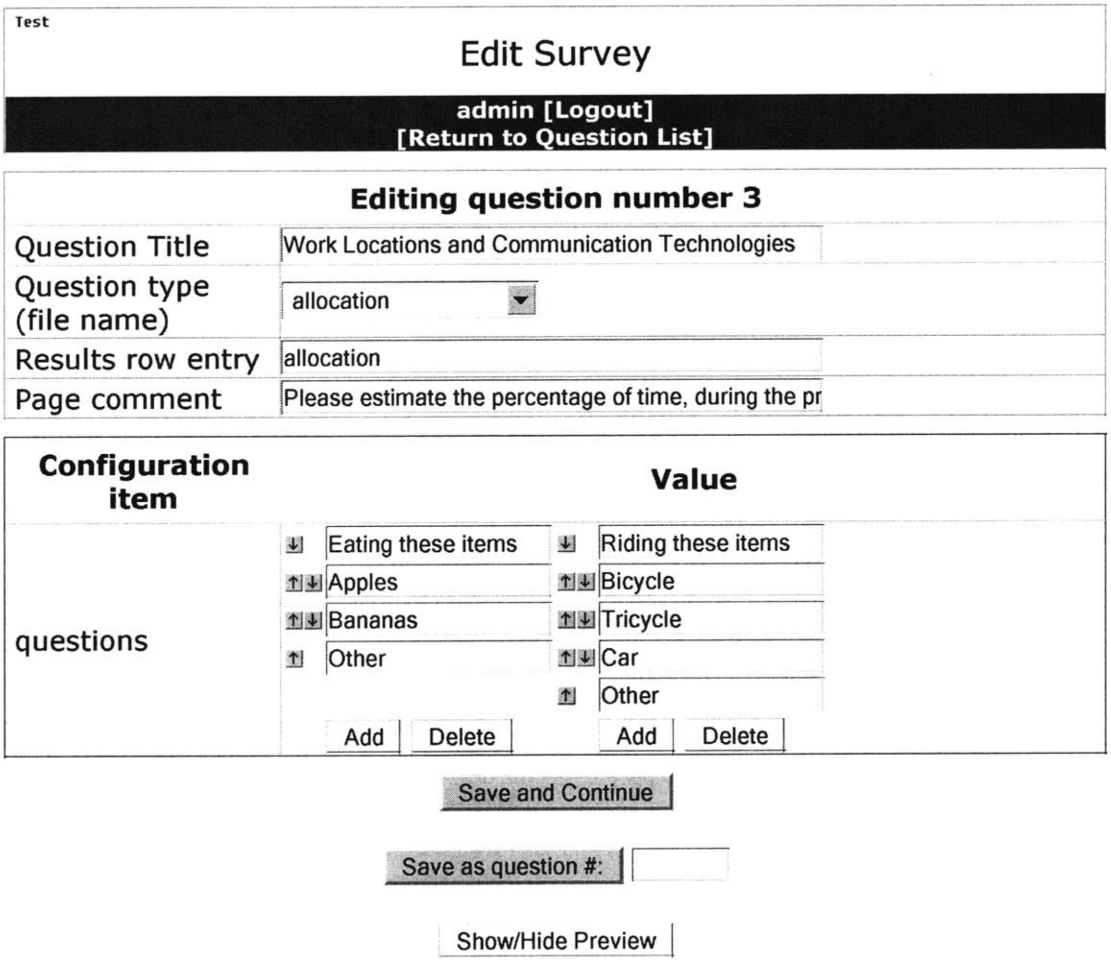


Edit Survey

Editing question number 1

Question Title	General Information
Question type (file name)	general_info
Results row entry	gen_information
Page comment	Please provide general information about your partici
Message	General Information
Extra settings	
Extra SQL	

Figure 3-4: "Edit Question" page for questions not using the `question_config` table



Test

Edit Survey

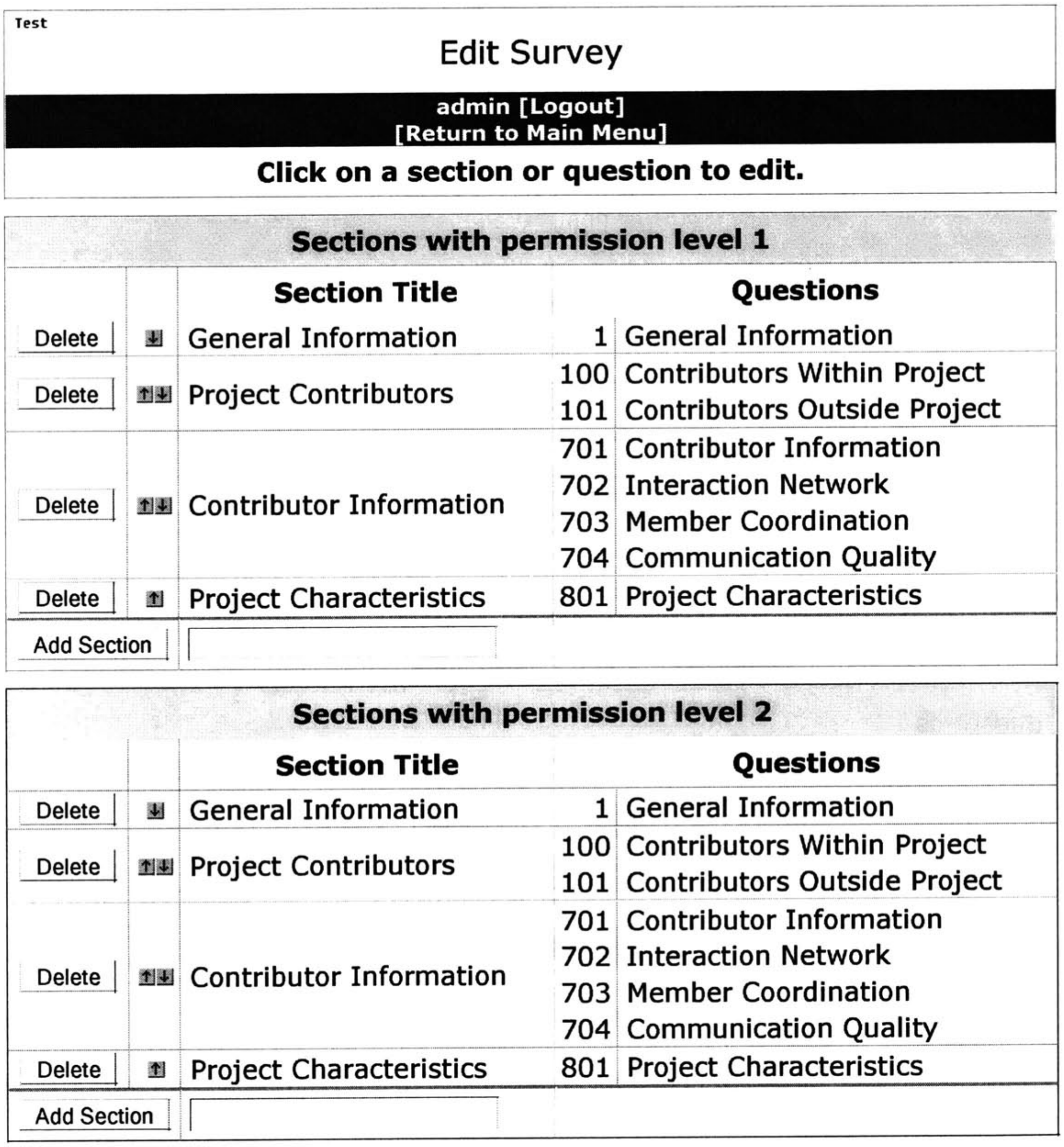
admin [Logout]
[Return to Question List]

Editing question number 3

Question Title	Work Locations and Communication Technologies
Question type (file name)	allocation
Results row entry	allocation
Page comment	Please estimate the percentage of time, during the pr

Configuration item	Value	
questions	Eating these items Apples Bananas Other	Riding these items Bicycle Tricycle Car Other

Figure 3-5: "Edit Question" page for questions that use the `question_config` table



Test

Edit Survey

admin [Logout]
[Return to Main Menu]

Click on a section or question to edit.

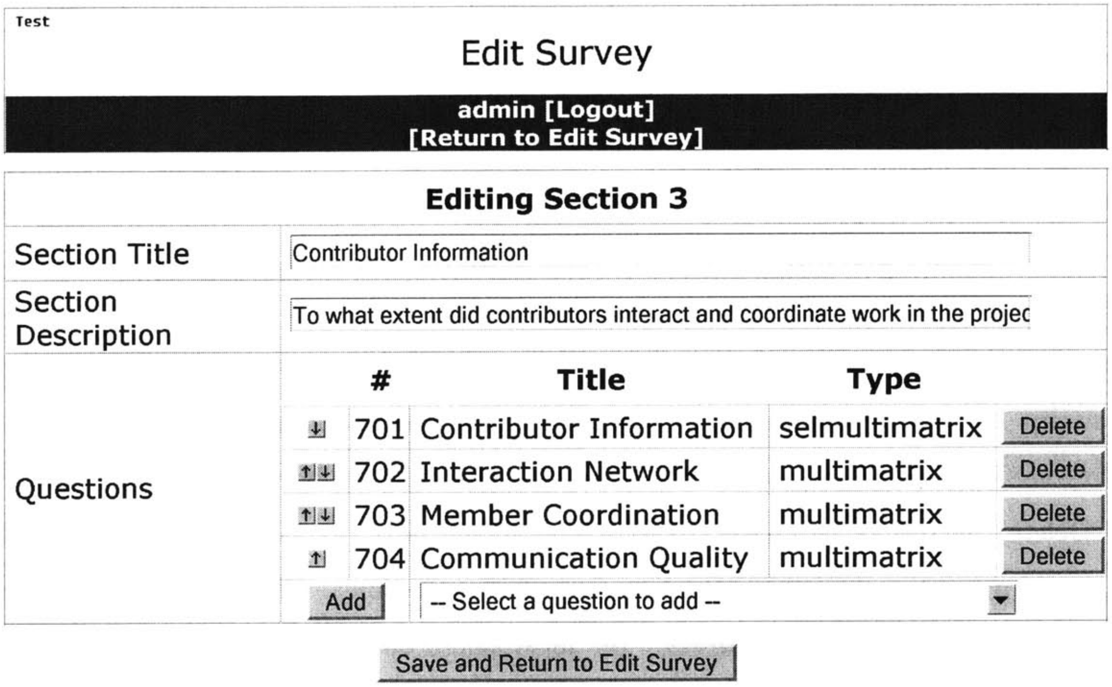
Sections with permission level 1

		Section Title	Questions	
Delete	↓	General Information	1	General Information
Delete	↑↓	Project Contributors	100 101	Contributors Within Project Contributors Outside Project
Delete	↑↓	Contributor Information	701 702 703 704	Contributor Information Interaction Network Member Coordination Communication Quality
Delete	↑	Project Characteristics	801	Project Characteristics
Add Section				

Sections with permission level 2

		Section Title	Questions	
Delete	↓	General Information	1	General Information
Delete	↑↓	Project Contributors	100 101	Contributors Within Project Contributors Outside Project
Delete	↑↓	Contributor Information	701 702 703 704	Contributor Information Interaction Network Member Coordination Communication Quality
Delete	↑	Project Characteristics	801	Project Characteristics
Add Section				

Figure 3-6: Survey Administration Console "Edit Survey" page



Test

Edit Survey

admin [Logout]
[Return to Edit Survey]

Editing Section 3

Section Title	Contributor Information
Section Description	To what extent did contributors interact and coordinate work in the projec

Questions

	#	Title	Type	
↓	701	Contributor Information	selmultimatrix	Delete
↑↓	702	Interaction Network	multimatrix	Delete
↑↓	703	Member Coordination	multimatrix	Delete
↑	704	Communication Quality	multimatrix	Delete
Add		-- Select a question to add -- ▾		

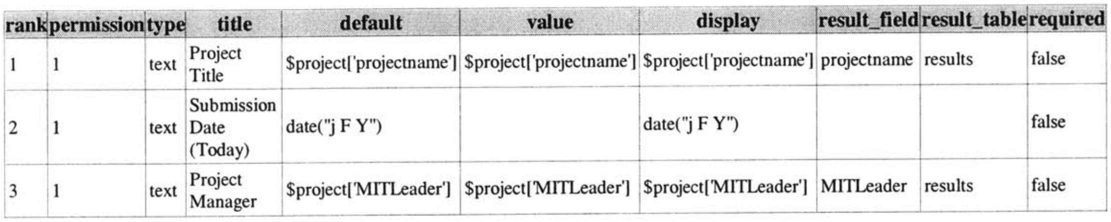
Save and Return to Edit Survey

Figure 3-7: Survey Administration Console "Edit Section" page

questions and sectionTitle from the question_sections table. The sections are first grouped by permission, then ordered by sectionID before being displayed to the user. For each section, the page displays: the section title, the questions contained by the section (question number and title), a delete button and reordering buttons. For each permission level, there is an "Add Section" button that can be used to create an empty survey section with the specified name.

Clicking on a question number or question title will bring the user to a question editing page as described above. Clicking on the "Delete" button deletes the corresponding table row from question_sections. Clicking on a reordering button will move a section up or down by swapping its sectionID with the section immediately before or after it in the table. Because the sectionID is used only for ordering and never for selecting specific survey sections, we have hidden the actual sectionID values and rely instead on a visual ordering of the rows.

Clicking on a section title sends the user to the "Edit Section" page. This page shows editable text boxes corresponding to the section title and section description. Questions contained by the section are displayed in the order they appear in the



rank	permission	type	title	default	value	display	result_field	result_table	required
1	1	text	Project Title	$project['projectname']	$project['projectname']	$project['projectname']	projectname	results	false
2	1	text	Submission Date (Today)	date("j F Y")		date("j F Y")			false
3	1	text	Project Manager	$project['MITLeader']	$project['MITLeader']	$project['MITLeader']	MITLeader	results	false

Figure 3-8: Sample rows from the `general_info_questions` table.

survey. The question number, question title and type of question are displayed for clarity. Questions can be reordered using iconic buttons in the same way as survey sections. A question can be removed from the section by using the appropriate "Delete" button. A pulldown at the bottom of the table of questions allows the user to add a question type from the list of available questions. When the user clicks on "Save and Return to Edit Survey," the form values on the page are used to update the corresponding row in the `question_sections` table.

3.3.3 Special Question Types

There are two survey pages that load their format from special database tables. The "Background Information" page (`person_info.php`) and the "General Information" question type (`general_info.php`) load from the `person_info_questions` and `general_info_questions` tables, respectively. These two pages are special because every item displayed by either page is configured by one table row. The function of the various fields is described below:

- `rank`: Ordinal value. Items are ordered by rank before being displayed.
- `permission`: Item is shown only if permission of user is equal to this value.
- `type`: Determines the way the item is visualized. Valid values are: input, select, text, blank, monthyear.
- `title`: This text is shown as a label to the left of the actual text or input element.

- `default`: If no value is found with which to populate this text field or input element, use this value by default.

- `value`: Use this value or sequence of values for the `value` attribute of the HTML input element.

- `display`: Use this value or sequence of values to determine what the user sees.

- `result_field`: When this question is submitted, write the user-determined value to this field.

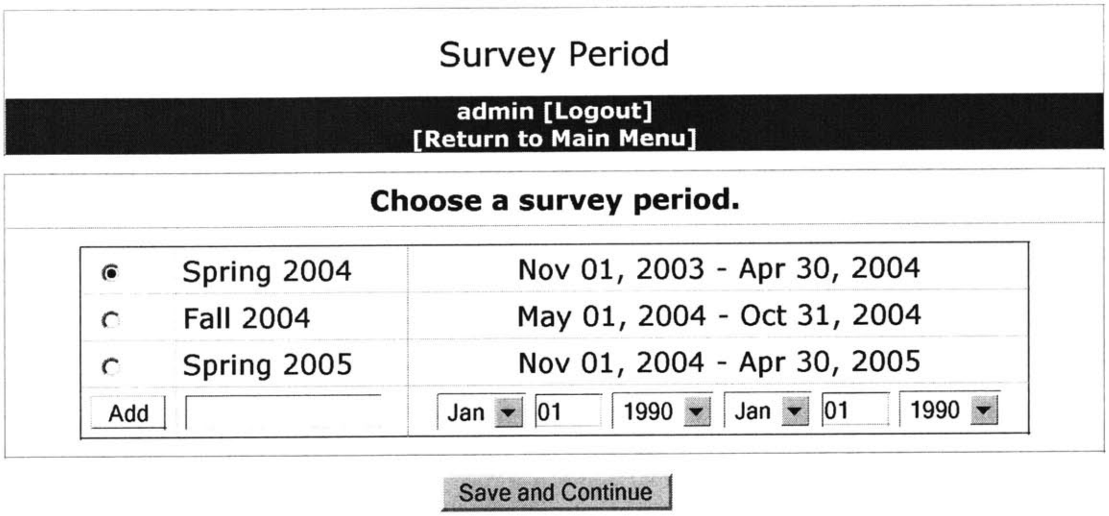
- `result_table`: When this question is submitted, write the user-determined value to this table.

- `required`: This boolean condition must evaluate to `false` in order to proceed to the next page. If it is optional for users to provide a value for this question, this field may simply contain `false`.

The values of these fields can be a mix of HTML, PHP and plain text. Due to their complex format, these tables are only updated by Project Assessment administrators who are familiar with the source code used to render these pages. As a result, the Survey Administration Console provides the "Special Question Types" page that allows the editing of these tables directly, without any simplifications. This functionality is implemented by the *surveyadmin.TableEdit* class, which exposes any database table to the user in the form of an interactive spreadsheet. This class is described further in section 3.3.5.

3.3.4 Survey Periods

Once a survey has been created, it can be distributed once or for multiple survey periods. Our hope is that most organizations will choose to continue with a survey over several periods so that IDI researchers can analyze time-varying data and discover performance trends. By default, surveys are initialized with a single period. Once all



Survey Period

admin [Logout]
[Return to Main Menu]

Choose a survey period.

●	Spring 2004	Nov 01, 2003 - Apr 30, 2004
○	Fall 2004	May 01, 2004 - Oct 31, 2004
○	Spring 2005	Nov 01, 2004 - Apr 30, 2005
Add		Jan ▾ 01 1990 ▾ Jan ▾ 01 1990 ▾

Figure 3-9: Selecting a Survey Period

results have been gathered from the first period, survey administrators can choose to create a second period and select it as active.

In the past, Project Assessment personnel have had to perform the transition from one period to the next by hand. There are several steps involved, and failing to perform any of them can cause problems that range from subtle to catastrophic. The "Survey Periods" page automates these steps as much as possible. This page loads table rows from the `periods` table and allows the user to select an existing period or create a new one. When creating a new period to use as the active one, the following steps must be performed:

1. Create a new row in the `periods` table with a period ID, description, start date and end date.

2. Append the new period ID to all rows in `people` that contain the current period ID in the `periods` field.

3. Copy all rows in the `projects` table with `periodID` set to the current period to new rows with `periodID` set to the new period.

4. Update all rows in the `user` table where `periodID` is equal to the current period; set it to the new period.

5. Set the value of the `periodID` setting to the new period ID.

If instead of creating a new period, the user wants to revert to an existing period, we can ignore all steps except the last two. Records in the `people`, `projects` and `periods` table already exist, so only the `user` table and `periodID` setting need to be changed.

3.3.5 Survey Settings

Survey settings in DPAS are stored in a `settings` database table as described in section 2.2.3. Since the format of the settings table is relatively simple, and users are only expected to need to edit one column (`value`), it makes sense to allow the table to be edited directly. As with the Background Information and General Information tables mentioned above, survey settings are edited by using the generic *surveyadmin.TableEdit* class.

The *surveyadmin.TableEdit* class is an example of an extension to *surveyadmin.AdminPage* that uses a combination of CSS and Javascript to create a more powerful user interface. Upon page load, *TableEdit* uses the value of the `table` parameter from the HTTP GET request as the name of the database table to display. If the `start` parameter is set, its value is used as the index of the first row to display (as an argument to the SQL `LIMIT` command). Once the proper rows are loaded from the database table, they are rendered as a two-dimensional table using standard HTML. Values which overflow the visible table cell can be viewed by hovering the mouse over the cell; the complete value will appear as a tooltip on modern web browsers.

When the user clicks on a table cell, a Javascript function `doEdit(cellID)` is used to make visible an HTML text area and align it with the top left corner of the selected cell. The text area is populated with the current value of the cell and selected for input focus. Once the user moves focus away from the text area (either by using the Tab key or by clicking outside of the area), the Javascript function `doHide()` is called to hide the text area and write the edited value back to the table cell. Cells whose values have been modified are highlighted to show that they differ from the

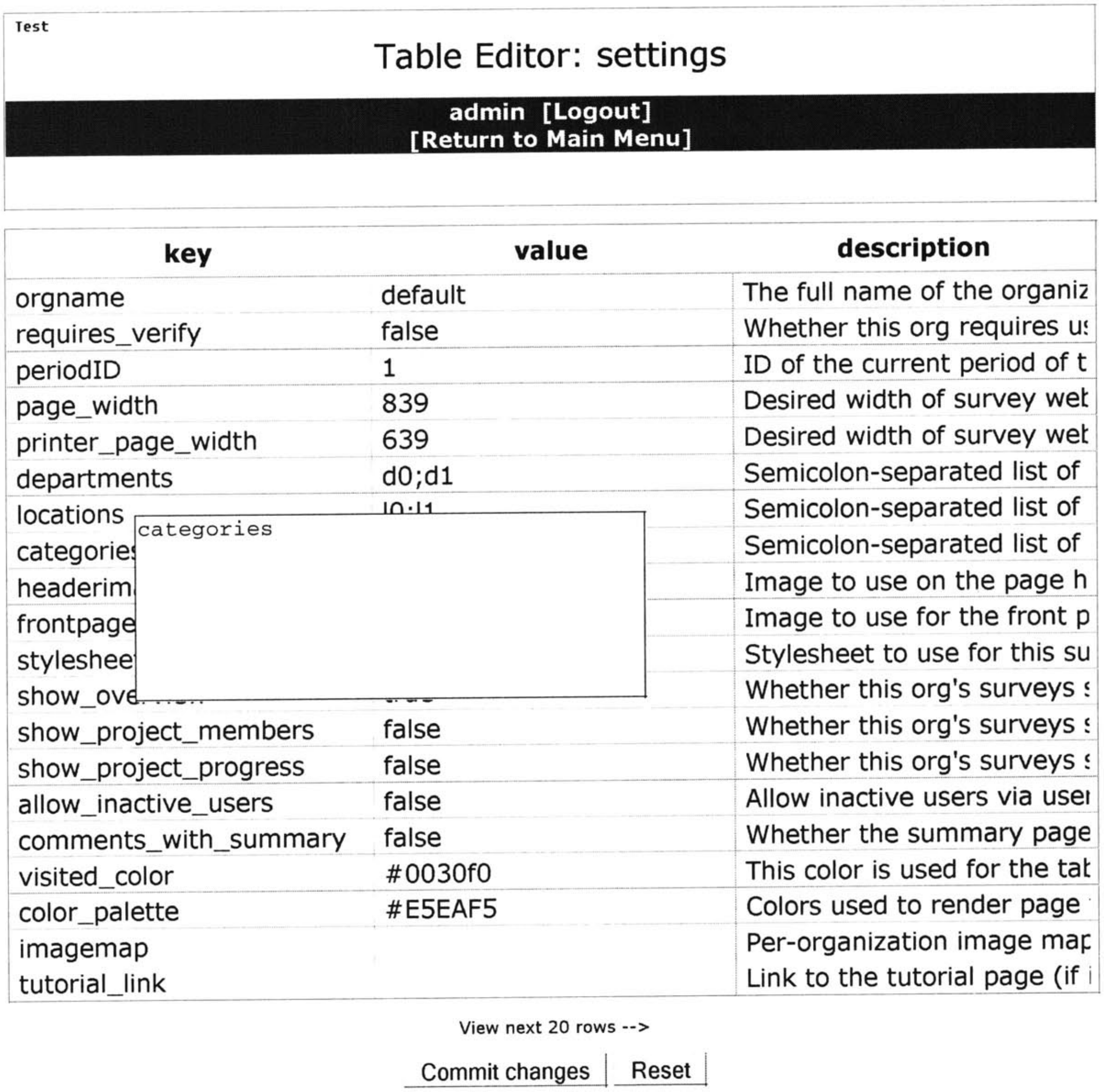


Table Editor: settings

key	value	description
orgname	default	The full name of the organiz
requires_verify	false	Whether this org requires us
periodID	1	ID of the current period of t
page_width	839	Desired width of survey wet
printer_page_width	639	Desired width of survey wet
departments	d0;d1	Semicolon-separated list of
locations	l0;l1	Semicolon-separated list of
categorie		Semicolon-separated list of
headerim		Image to use on the page h
frontpage		Image to use for the front p
styleshee		Stylesheet to use for this su
show_ove	true	Whether this org's surveys s
show_project_members	false	Whether this org's surveys s
show_project_progress	false	Whether this org's surveys s
allow_inactive_users	false	Allow inactive users via usei
comments_with_summary	false	Whether the summary page
visited_color	#0030f0	This color is used for the tat
color_palette	#E5EAF5	Colors used to render page
imagemap		Per-organization image map
tutorial_link		Link to the tutorial page (if i

categories

Figure 3-10: The table editing GUI in action

values currently stored in the database. The user can click "Commit changes" to write modified values back to the database, or "Reset" to discard changes and refresh the table from the database.

3.4 Extending the Console

Each Survey Administration Console page is implemented as a single class that extends the *surveyadmin.AdminPage* base class. This base class provides two types of methods that can be used by dependent classes: stub methods that subclasses should implement in order to deliver specialized behavior, and basic *AdminPage* methods that subclasses should have no reason to reimplement. First, let us examine the methods that do not require implementation by subclasses. For the purpose of clarity I will use a pseudo-PHP syntax that makes types explicit.

- `AdminPage.initialize(NavTree $t)` This method registers the current page or class with the provided navigational tree object.

- `AdminPage.getPageInfo(String $param)` If `$param` is specified, this method examines the current object for an entry with key `$param` in the private `pageinfo` array, and returns the value if found or the empty string if not. If `$param` is not specified, it returns the entire `pageinfo` array.

- `AdminPage.getUserBar()` This method returns an HTML string that is shown at the top of the page under the title bar. By default it displays the user's name and a link that the user can click to log out of the console.

- `AdminPage.showHeader(String $headerURL, String $returnlink)` Renders the header section of this page given a header image URL and a navigational link allowing the user to return to the previous page. The page title is loaded from the `'name'` entry in the private `pageinfo` array.

- `AdminPage.showFooter()` Renders the footer section of this page. In its current

Listing 3.1: Javascript functions used by Table Editor.

```
function doEdit(iRow, sKey) {
  // retrieve table cell with ID = table_<iRow>_<sKey>
  cellID = 'table_'+iRow+'_'+sKey;
  cell = document.getElementById(cellID);
  // get reference to cell editor
  editing = document.getElementById('editing');
  editing.value = cellID;
  editor = document.getElementById('editpad');
  // set value of cell editor to value of table cell
  editor.value = cell.lastChild.value;
  // make cell editor visible
  editor.style.display = '';
  // get abolute position of cell using getLeft() and getTop()
  // position cell editor using CSS absolute positioning
  editor.style.left = getLeft(cell);
  editor.style.top = getTop(cell);
  // set input focus to cell editor
  editor.focus();
}

function doHide() {
  // get reference to cell editor
  editor = document.getElementById('editpad');
  editing = document.getElementById('editing');
  // determine the table cell currently being edited
  cell = document.getElementById(editing.value);
  // if value differs from original:
  if (cell.lastChild.value != editor.value) {
    // save value of editor back to table cell
    cell.firstChild.data = editor.value;
    cell.lastChild.value = editor.value;
    // highlight cell using CSS
    cell.style.backgroundColor = '#E5EAF5';
    // call doSetModified(cellID)
    doSetModified(editing.value);
  }
  // hide cell editor
  editor.style.display = 'none';
}
```

incarnation, the Survey Administration Console does not use a footer section; this method is a stub.

- `AdminPage.linkParams(String $url, Array $value)` This utility method assumes that `$value` is an associative array. It iterates through the key-value pairs, attaches them to the URL in the form of HTTP GET parameters and returns the resulting string.

The above methods should very rarely require extension, in contrast with the following stub methods that should always be implemented by subclasses:

- `[Constructor](String $orgID)` Every subclass should provide a constructor that calls `parent::AdminPage($orgID)` and then sets appropriate values in the associative array `$this->pageinfo`. Initialization that does not depend on HTTP GET or POST parameters or PHP session variables should also occur here.

- `AdminPage.checkDisplay(Array $get, Array $post, Array $session)` This method should inspect the HTTP GET parameters, POST values and PHP session associative array, and should return `true` if and only if the prerequisite conditions for displaying this page are met.

- `AdminPage.getJavascript()` This function should return all of the Javascript code required by this class as a string. The Javascript code should NOT be enclosed by `<script></script>` tags.

- `AdminPage.getCSS()` This function should return any page-specific CSS styling. This CSS should NOT be enclosed by `<style></style>` tags.

- `AdminPage.showContent(Array $get, Array $post, Array $messages)` This method, in conjunction with `getJavascript()` and `getCSS()`, should perform most of the "heavy lifting" of the class. It should render HTML directly (e.g. by using `echo "<html>";`) based on the HTTP GET and POST parameters provided.

- `AdminPage.validate(Array $get, Array $post)` This method should perform validation on the contents of any input elements generated by the `showContent()` method. Failed validation should be indicated by returning an array containing one or more messages to display to the user. Successful validation should be indicated by returning an array with zero elements. If a subclass does not need to perform any validation, it can simply rely on the default *AdminPage* implementation.

- `AdminPage.submit($get, $post)` This method can assume that input validation has succeeded. It should perform any updates to system state implied by user input.

Each subclass of *surveyadmin.AdminPage* represents a web page, but we still need some support logic to control page display and handle page transitions. For that reason, the PHP pages `admin_page.php` and `admin_page_submit.php` implement a basic event loop that displays an administrative page to the user, then handles user input and forwards control to the next page.

Chapter 4

Conclusion

Working on the Project Assessment survey system was a unique and interesting challenge. As an undergraduate and graduate student of computer science at MIT, much of my coursework has focused on the design and implementation of software systems to solve well-defined problems. Most of the time the system in question is subject to constraints in the form of environmental limitations (operating system, hardware capabilities) or user specifications. In this case, I was given an existing system to work on, with its own assumptions and design decisions. At this point there are basically two options for the developer. One is to design a new solution from scratch around the current and anticipated set of requirements, and then reuse as much as possible from the old system. The other, more realistic path is to consider the properties of the existing system as restrictions in and of themselves, and redesign or reimplement within those restrictions as well as possible. This second option is really the **only** option when the existing system is in daily use, as is the case with Project Assessment. The following section describes some of the challenges I experienced over the course of the project, and some of the solutions I applied.

4.1 Challenges Overcome

As with all engineering projects, there were both technical and non-technical challenges to overcome. The first challenge I encountered when I signed on to the project was the task of overcoming the steep learning curve of the Project Assessment system. When I joined the project, the most comprehensive documentation available was Fumiaki's thesis. By the time I started working, however, it had been over half a year since that document was written, and there had been several minor and major changes in the interim. In addition, due to the rapid development cycle of the project, the source code was rather sparsely commented. Finally, though I had had experience with other scripting languages such as Perl, I had never performed any programming in PHP. All of these factors made the first few weeks rather intimidating, but with help and guidance from the other Project Assessment developers, I was able to get up to speed fairly quickly. I have done my best to streamline the experience for the next Project Assessment developer by adding inline comments where possible, external documentation where necessary, and by pruning obsolete or extraneous source code anywhere it appears.

PHP is a unique language to use for web development because it resembles both a traditional programming language as well as a templated HTML generation language. For example, arbitrary external documents can be included using the PHP `include()` or `include_once()` functions to provide additional functionality or to generate content directly. This increased flexibility comes at the cost of code not knowing the context in which it is being executed; relative pathnames become a problem if included source code resides in a different directory, for example. This was particularly a problem when writing object-oriented code, since PHP class files must be included by filename rather than by class name. I solved this problem by unifying access to classes via `class_utils.php`, which was at a fixed path relative to all class source code files. Object-oriented source code now only needs to know the path to `class_utils.php` rather than to each individual class file. Classes can then be included by using the provided method `include_class(String $classname`, where `$classname` is a period-delimited class name like *surveyadmin.AdminPage*. Class name segments denote packages, which are mapped directly onto filesystem directories. For example, the definition of *surveyadmin.AdminPage* is located in `/classes/surveyadmin/AdminPage.php` in the root directory of the Project Assessment web server.

Object-oriented design principles were used to build persistent data objects that simplify access to system state. By abstracting away the details of how survey settings and text strings are stored and retrieved, I have managed to simplify survey code that requires knowledge of the survey settings and strings. A second consequence is that it is now possible to write a drop-in replacement for the survey settings implementation that uses a different storage medium. If we ever wanted to revert to a file-based settings repository, the *Settings* class can be rewritten without affecting the rest of the survey code at all.

I also succeeded at writing an object-oriented page framework for the Survey Administration Console. It is my hope that this model can be applied to the existing survey pages as well. By relying on base classes to provide common code and requiring subclasses to implement only specific functionality, the size of the overall code base can be reduced, and the code for each individual page can be much simpler.

4.2 Lessons Learned

Over the course of the project, I have learned several important lessons about software development in a cooperative environment. These lessons are recorded here so that future Project Assessment developers may glean some benefit from my experiences.

4.2.1 Collaboration

One of the most exciting aspects of working on the Dynamic Project Assessment System was the fact that it was part of a much larger picture; DPAS is merely one component of the overall Initiative for Distributed Innovation. Professor Cummings held IDI lunch meetings every two weeks over the course of the spring semester so that all of the participants could see how our individual pieces interdepend and interact. These meetings were crucial because they allowed us to brainstorm together and rectify mismatched expectations or assumptions. They were also a source of motivation, since I was able to see how my work played a role in the larger system.

4.2.2 Time Management

Considering the busy schedule of the project and the limited time available, I learned very quickly how to manage my time effectively. On average, I would spend about two-thirds of each day on features, bug fixes or modifications directly or indirectly related to active surveying organizations. I spent the remainder of the time cleaning up the project source code, designing the Survey Administration Console, writing documentation or redesigning survey system components.

4.2.3 Development vs. Deployment

When working on a system that is deployed in a production environment, it is very important that the system remain stable and available from the perspective of its users. To that end, it is most desirable for changes to the codebase to propagate atomically and invisibly to the production environment. Although there is currently no way for us to guarantee atomic, transactional updates to the server, committing changes to any particular file typically takes only a few seconds. In spite of the very limited impact of any single update on the availability of the Project Assessment server, most updates occur when the primary users of the survey system (researchers in the United States and United Kingdom) are least likely to be using it. Large, complex or fundamental updates are performed over the weekend, since in these cases the likelihood of error is higher and bugs can be fixed performed before they negatively affect the experience of any users.

4.3 Future Directions

Project Assessment continues to evolve on a daily basis. At the time of writing, we are preparing to launch an offsite installation of the survey system at a new organization. There are several new features in the pipeline that will need to be implemented before the final launch; these new features will in turn make the system more powerful and (hopefully) attract additional organizations. The sections below describe a few of the short- and long-term goals for the Project Assessment system.

4.3.1 Web Page Design

Although significant progress has been made in migrating web page style information to CSS, there is more to be done. In the eventual ideal case, in-line styling via the STYLE attribute would be eliminated completely. This is unlikely due to the constantly evolving look of the site, but it should be possible to enable multiple media-dependent views via stylesheets. Once this has been accomplished for one medium, others should follow with a minimum of effort. The first media-specific stylesheet developed should be the one that implements the "Printer Mode" view of the survey pages.

4.3.2 Architectural Improvements

I envision several architectural enhancements to the Project Assessment system that will make the system much easier to maintain and extend. First, survey pages should be rewritten according to an object-oriented framework similar to the one used by the Survey Administration Console. This will immediately reduce the amount of work required to maintain the survey pages, since in the current system functionality is duplicated from page to page. Changing the behavior of one item common to multiple survey pages currently requires changing all copies of the item. Bugs can arise when a copy slips through the cracks.

The benefits derived from an object hierarchy will be magnified if HTML generation is performed using PHP's implementation of the W3C Document Object Model specification. Currently, subclasses in the object hierarchy of the Survey Administration Console that want to reimplement HTML-generating methods must provide HTML that completely replaces that generated by the parent methods. If DOM trees were used to represent HTML, subclass methods could retrieve HTML from the parent class in mutable form, perform minor changes and then return the resulting HTML for display. Utilizing PHP's DOM implementation in a portable manner will require upgrading to PHP 5.

Although survey settings and strings have been encapsulated in data objects, most other operations on survey data are performed with raw SQL queries. In some cases the SQL syntax is very intuitive, but when selecting from multiple tables or using multiple conditions, SQL can quickly become unreadable. Using SQL queries directly also creates a strong dependence on the existing database schema. This can make redesigning the data architecture very painful, even if the goals of the system are no longer reflected by the current structure of the database. Limiting the number of code paths that access the database directly by encapsulating those functions in data objects will therefore make the overall system more amenable to change.

In the current implementation, the flow of control from one survey page to the next is internalized in the source code of the survey pages themselves. Although this is convenient from the perspective of the programmer of a particular survey page, it makes the overall page flow more difficult to discern. The Project Assessment survey flow has become complex enough that it would be better represented in a form external to the survey page source code. By no small coincidence, an external page flow definition is one property common among all web development frameworks based on the Model-View-Control paradigm. As the Project Assessment system becomes more and more complex, the logical compartmentalization of model, view and control components provided (and required) by a MVC framework becomes more and more enticing. In the long term, the Project Assessment system should be adapted to run on top of a framework like Mojavi, php.MVC or Studs [1].

4.3.3 Open Source Release Work

In order for the Project Assessment system to be usable as an open-source survey distribution platform, much more documentation needs to be written. In particular, we need:

1. A step-by-step guide to writing new survey question types. I learned by studying examples, but I could have saved considerable time had there existed adequate documentation.

2. A guide to configuring a survey's look and feel. This primarily involves writing a CSS stylesheet, writing the HTML for a page header, and choosing a logo image.

3. A document listing design decisions and the reasoning behind them. This thesis should suffice for the case of the Dynamic Project Assessment System, but the primary document should be kept up to date as the project moves forward.

I anticipate that I will write draft versions of all three docments before I hand development over to the next Project Assessment development lead.

4.4 Final Words

This project, in a sense, has been a microcosm of everything I've ever learned about software engineering. All of the classic programming obstacles and system design impediments were present, but I have found that my education here has equipped me well to handle the challenges involved. In the end, I feel fortunate to have had the opportunity to work on a system that will not languish in electronic limbo, but will be actively used for some time to come.

Appendix A

Source Code Examples

Listing A.1: *Settings* class source code.

```
/* Settings.php
  Created 03/07/05 by birenroy

  Loads settings from the "settings" table in this org database.
     */

class Settings {
  var $org = "";
  var $ary = array();
  var $connection;

  // Constructor
  function Settings($orgID) {
    $this->org = $orgID;
    $this->connection = mysql_connect('localhost', 'root', '');
    mysql_select_db($this->org, $this->connection);
    $this->loadSettings();
  }

  // Returns an array of default values. These values are used if
  // a value does not already exist in the 'settings' table for
  // this organization. See checkSettings() for more details.
  function getDefaults() {
    $defaults = array(
      // key => array(default value, description)
      ...
    );
    return $defaults;
```

```
}

// Returns the value of the setting '$key'
function get($key) {
  if (array_key_exists($key, $this->ary))
    return $this->ary[$key];
  else
    return "";
}

// Returns the boolean value of the setting '$key'
function getFlag($key) {
  if (array_key_exists($key, $this->ary))
    return !strcmp(strtolower($this->ary[$key]), "true");
  else
    return false;
}

// Returns a setting as an URL with base "$_SESSION['org']/"
function getURL($key) {
  return $this->org . "/" . $this->get($key);
}

// Sets the value for a particular key, with an optional
    description.
function set($key, $value, $description="") {
  $c = $this->connection;
  if (array_key_exists($key, $this->ary)) {
    $dsql = !empty($description) ? ", description='{
       $description}'" : "";
    $sql = "UPDATE settings SET value='{$value}' {$dsql} WHERE
       `key`='{$key}'";
  } else {
    $sql = "INSERT INTO settings VALUES ('{$key}', '{$value}',
       '{$description}');";
  }
  mysql_query($sql, $c);
}

// Checks for the existence of a 'settings' table for this
// organization. Creates one if it doesn't exist yet. If there
// is no row corresponding to a particular key, one is created
// from the default values provided by getDefaults().
function checkSettings() {
  $c = $this->connection;
```

```
    $sql = "CREATE TABLE IF NOT EXISTS `settings` (
      `key` varchar(64) NOT NULL,
      `value` text NOT NULL,
      `description` text NOT NULL,
      KEY `key` (`key`)
      ) TYPE=MyISAM COMMENT='Survey settings.'";
      mysql_query($sql, $c)
      or die("Unable to create settings table!: ".$sql."\n".
         mysql_error($c));
    $sql = "SELECT `key` FROM settings";
    $res = mysql_query($sql, $c)
      or die("Query failed!: ".$sql);

    $shouldexist = $this->getDefaults();
    while ($row = mysql_fetch_assoc($res)) {
      unset($shouldexist[$row['key']]);
    }
    foreach ($shouldexist as $key => $row) {
      $value = addslashes($row[0]);
      $description = addslashes($row[1]);
      $sql = "INSERT INTO settings SET `key`='{$key}', value='{
         $value}',
        description='{$description}'";
      mysql_query($sql, $c)
        or die("Insert failed!: ".$sql);
    }
  }

  // Loads settings from the database after running checkSettings
     ().
  function loadSettings() {
    $this->checkSettings();
    $c = $this->connection;
    $sql = "SELECT * FROM settings";
    $res = mysql_query($sql, $c)
      or die("Query failed!: ".$sql);
    while ($row = mysql_fetch_assoc($res)) {
      $this->ary[$row['key']] = $row['value'];
    }
  }

}
```

Listing A.2: *Strings* table source code.

```
/* Strings.php
  Created 03/07/05 by birenroy

  Loads strings from the "strings" table in this org database. */

class Strings {
  var $org = "";
  var $ary = array();
  var $connection;

  // Constructor
  function Strings($orgID) {
    $this->org = $orgID;
    $this->connection = mysql_connect('localhost', 'root', '');
    mysql_select_db($this->org, $this->connection);
    $this->checkStrings();
    $this->loadStrings();
  }

  function getDefaults() {
    $defaults = array(
      // label => array(default string, description)
      ...
    );
    return $defaults;
  }

  // Returns the value of the setting '$key'
  function get($key) {
    $str = "";
    if (array_key_exists($key, $this->ary))
      $str = $this->ary[$key];
    if (strpos($str, "$"))
      $str = eval("return \"{$str}\";");
    return $str;
  }

  // Loads strings from the DB
  function loadStrings() {
    $c = $this->connection;
    $sql = "SELECT * FROM strings";
    $res = mysql_query($sql, $c)
      or die("Query failed!: ".$sql);
    while ($row = mysql_fetch_assoc($res)) {
      $this->ary[$row['label']] = $row['string'];
    }
  }

  function checkStrings() {
    $c = $this->connection;
    $sql = "CREATE TABLE IF NOT EXISTS `strings` (
      `stringID` int(11) NOT NULL auto_increment,
      `label` varchar(64) NOT NULL,
      `string` text NOT NULL,
      `description` text NOT NULL,
      KEY `stringID` (`stringID`)
      ) TYPE=MyISAM COMMENT='Strings used throughout the survey
        system.'";
      mysql_query($sql, $c)
      or die("Unable to create strings table!: {$sql}\n" .
        mysql_error($c));
    $sql = "SELECT label FROM strings";
    $res = mysql_query($sql, $c)
      or die("Query failed!: ".$sql);

    $shouldexist = $this->getDefaults();
    while ($row = mysql_fetch_assoc($res)) {
      unset($shouldexist[$row['label']]);
    }
    foreach ($shouldexist as $label => $row) {
          $str = addslashes($row[0]);
            $description = addslashes($row[1]);
      $sql = "INSERT INTO strings SET label='{$label}', string='{
        $str}',
        description='{$description}'";
      mysql_query($sql, $c)
        or die("Insert failed!: ".$sql);
    }
  }

}
```

Listing A.3: *surveyadmin.AdminPage* base class.

```php
/*  AdminPage.php
  Created 04/25/05 by birenroy

  Class used to display/update a surveyadmin console page. */

class AdminPage {
  var $org;
  var $connection;
  var $pageinfo;

  // Constructor. Subclasses should extend this method by calling
  // "parent::AdminPage($orgID)" and then setting their own
  // $pageinfo keys.
  function AdminPage($orgID) {
    $this->org = $orgID;
    $this->connection = mysql_connect('localhost', 'root', '');
    mysql_select_db($this->org, $this->connection);
    $this->pageinfo = array(
      'name' => "Generic Admin Page",
      'id' => "surveyadmin.AdminPage",
      'error_target' => "surveyadmin.AdminPage",
      'failed_display_check' => "surveyadmin.AdminPage"
    );
  }

  //-----------------------------------------------------------------
  // Base class methods that DO NOT need to be implemented by
  // subclasses.

  // Initializes this page for display.
  function initialize($navtree) {
    $navtree->register_page($this->getPageInfo('name'), $this->
        getPageInfo('id'));
  }

  // Returns the value of a pageinfo parameter given the
      parameter
  // name. Returns the entire associative array if the parameter
  // name is blank.
  function getPageInfo($param="") {
    if (!empty($param))
      if (isset($this->pageinfo[$param]))
        return $this->pageinfo[$param];
      else
```

```
        return "";
    else
      return $this->pageinfo;
  }

  // Subclasses should only implement if they want to use a
     different
  // user display bar.
  function getUserbar() {
    return "<tr><td class='userdisplay' align='center'>
      <font class='userdisplay'>{$_SESSION['admin']['username']}&
         nbsp;
      <a class='returnlink' style='color: inherit;' href='logout.
         php'>[Logout]</a></font>
    </td></tr>";
  }

  // This function should render the header HTML directly.
     Subclasses
  // should not have to implement this method.
  function showHeader($headerURL="", $returnlink="") {
    $imagefile = realpath("../" . $headerURL);
    $imageinfo = getimagesize($imagefile);
    $height = $imageinfo[1] > 50 ? $imageinfo[1] : 50;
    $messages = isset($_SESSION['messages']) ? $_SESSION['
       messages'] : array();
    unset($_SESSION['messages']);
?>
  <div class='header'>
  <table cellpadding='0' cellspacing='0'>
    <!-- Display header line on top of the page -->
    <tr><td align='center' style='height: <?= $height ?>;
       background-repeat: no-repeat; background-image: url
       ("../<?= $headerURL ?>");'>
      <span style='font-size: 120%;'><?= $this->getPageInfo('name
         ') ?></span>
    </td></tr>
    <?= $this->getUserbar() ?>
    <tr><td class='userdisplay' align='center'>
      <?= $returnlink ?>
    </td></tr>
    <tr><td class='subtitle'><?= $this->getPageInfo('subtitle')
       ?></td></tr>
    <tr><td class='errors' style='color: #ff0000; font-weight:
       bold;'><?= implode("<br>", $messages) ?></td></tr>
  </table>
  </div>
<?php
  }

  // This function should render footer HTML directly. Subclasses
  // should not have to implement this.
  function showFooter() {
    echo "";
  }

  function linkParams($url, $params) {
    if (empty($params))
      return $url;
    $first = true;
    $str = $url;
    foreach ($params as $key => $value) {
      $str .= ($first ? "?" : "&") . $key . "=" . $value;
      $first = false;
    }
    return $str;
    }

  //--------------------------------------------------------------------
  // Base class methods that SHOULD be overridden by subclasses.

  // Subclasses should return 'false' if the prerequisite
  // conditions for this page to display have not been met.
  function checkDisplay($get, $post, $session) {
    return true;
  }

  // Subclasses should return a string containing all of the
  // javascript required by this administrator page.
  function getJavascript() {
    return "";
  }

  // Subclasses should return a string containing all of the CSS
  // required by this administrator page.
  function getCSS() {
    return "";
  }

  // This function should render content HTML directly.
  function showContent($get, $post, $messages) {
?>
  <div class='content'>
    <span class='alert'><?= implode("<br>", $messages) ?></span>
  </div>
  <div class='content'>
  <table>
    <tr><td>Here is some sample content!</td></tr>
  </table>
  </div>
  <div class='content'>
  <table>
    <tr><th>Keys</th><th>Values</th></tr>
    <tr><td>name</td><td><?= $this->getPageInfo('name') ?></td></
       tr>
    <tr><td>ID/classname</td><td><?= $this->getPageInfo('id')
       ?></td></tr>
  </table>
  </div>
<?
  }

  // Subclasses should implement this method to perform form
  // validation. The function should return an array of messages
  // that will be displayed to the user. If the array contains
  // zero elements, validation is assumed to have succeeded.
  // This method SHOULD NOT modify $_GET, $_POST, $_SESSION.
  function validate($get, $post) {
    return array();
  }

  // Subclasses should implement this method to handle form
  // submissions. This method can assume that the validation
  // method has succeeded.
  function submit($get, $post) {
  }

}
```

Listing A.4: *surveyadmin.TableEdit* extension of *surveyadmin.AdminPage.*

```
/*  TableEdit.php
  Created 04/25/05 by birenroy

  Implements the Table Editor for the Administrative Console. */

include_class("surveyadmin.AdminPage");

class TableEdit extends AdminPage {
  var $rowsPerPage = 20;

  function TableEdit($orgID) {
    parent::AdminPage($orgID);
    $this->pageinfo = array(
      'name' => "Table Editor",
      'id' => "surveyadmin.TableEdit",
      'subtitle' => "Click on a table cell to edit it.",
      'error_target' => "surveyadmin.TableEdit",
      'failed_display_check' => "surveyadmin.MainPage"
    );
  }

  // Returns 'false' if the prerequisite conditions for this page
      to display
  // have not been met.
  function checkDisplay($get, $post, $session) {
    $tablename = "";
    if (!empty($get['table']))
      $tablename = $get['table'];
    else if (!empty($post['table']))
      $tablename = $post['table'];
    $sql = "SELECT * FROM {$tablename} LIMIT 1";
    $res = mysql_query($sql);
    // return true iff table exists for this database
    return ($res !== false);
  }

  // Returns a string containing all of the javascript required
     for this page.
  function getJavascript() {
    return "
    function doEdit(iRow, sKey) {
      cellID = 'table_'+iRow+'_'+sKey;
      cell = document.getElementById(cellID);
      editing = document.getElementById('editing');
    editing.value = cellID;
    editor = document.getElementById('editpad');
    editor.value = cell.lastChild.value;
    editor.style.display = '';
    editor.style.left = getLeft(cell);
    editor.style.top = getTop(cell);
    editor.focus();
  }
  function doHide() {
    editor = document.getElementById('editpad');
    editing = document.getElementById('editing');
    cell = document.getElementById(editing.value);
    if (cell.lastChild.value != editor.value) {
      cell.firstChild.data = editor.value;
      cell.lastChild.value = editor.value;
      cell.style.backgroundColor = '#E5EAF5';
      doSetModified(editing.value);
    }
    editor.style.display = 'none';
  }
  function doSetModified(cellID) {
    key = cellID.split('_');
    key = key[1];
    modified = document.getElementById('modified');
    if (modified.value.search(','+key+',') == -1) {
      modified.value = modified.value + key + ',';
    }
  }
  function getLeft(oNode) {
    var iLeft = 0;
    while(oNode.tagName.toLowerCase() != 'body') {
      iLeft += oNode.offsetLeft;
      oNode = oNode.offsetParent;
    }
    return iLeft;
  }
  function getTop(oNode) {
    var iTop = 0;
    while(oNode.tagName.toLowerCase() != 'body') {
      iTop += oNode.offsetTop;
      oNode = oNode.offsetParent;
    }
    return iTop;
  }";
}
```

```
// Returns a string containing all of the CSS required for this
   page.
function getCSS() {
  return "
  div.main {
    width: 85%;
    max-width: 85%;
  }
  table.content {
    table-layout: fixed;
  }
  table.content th {
    font-size: 90%;
    overflow: hidden;
  }
  table.content td.cell {
    font-size: 90%;
    padding: 1px 5px;
    border: 1px solid #a0a0a0;
    border-width: 1px 0px 0px 1px;
    white-space: nowrap;
    overflow: hidden;
  }
  table.content td.cell:first-child {
    border-left-width: 0px;
  }";
}

// This function should render content HTML directly.
function showContent($get, $post, $messages) {
  $tablename = $get['table'];
  $sql = "SELECT * FROM {$tablename} ";
  $startrow = !empty($get['start']) ? $get['start'] : 0;
  $endrow = $startrow + $this->rowsPerPage;
  $sql .= "LIMIT {$startrow}, {$endrow}";
  $res = mysql_query($sql, $this->connection);
  $rows = array();
  if (false !== $res) {
    while (false !== ($row = mysql_fetch_assoc($res)))
      $rows[] = $row;
  }
  if (count($rows) == 0) {
    if ($startrow == 0) {
        echo "<div class='content' style='text-align: center;'><
           span style='font-weight: bold;'>No database rows found
            in table {$tablename}!</span></div>";
        return ;
      } else {
        $_GET['start'] = ($startrow < $this->rowsPerPage) ? 0 :
           $_GET['start'] - $this->rowsPerPage;
        $params = $this->join($_GET);
        echo "<div class='content' style='text-align: center;'>
           No additional rows. <a href='admin_page.php?{$params
           }'>Return to previous rows</a>.</div>";
        return ;
      }
    }
?>
  <textarea id='editpad' style='display: none; position: absolute
     ;' cols='30' rows='4' onblur='doHide();return false;'></
     textarea>
  <input type='hidden' name='table' value='<?= $tablename ?>'>
  <input type='hidden' id='editing' value=''>
  <input type='hidden' id='modified' name='modified' value=','>
  <input type='hidden' name='keys' value='<?= implode(",",
     array_keys($rows[0])) ?>'>
  <div class='content'>
    <table class='content' cellspacing='0' cellpadding='0' width=
       '100%'>
<?php
    echo "<tr><th>" . implode("</th><th>", array_keys($rows[0]))
       . "</th></tr>\n";
    foreach ($rows as $i => $row) {
      echo "<tr>";
      foreach ($row as $key => $value)
        echo $this->renderCell($i+$startrow, $key, $value);
      echo "</tr>\n";
    }
?>
    </table>
  </div>
  <div style='padding-top: 10px;' width='100%'>
<?php
    $start = $_GET['start'];
    $_GET['start'] = $start - $this->rowsPerPage;
    $backparams = $this->join($_GET);
    $_GET['start'] = $start + $this->rowsPerPage;
    $params = $this->join($_GET);
    if ($startrow > 0)
      echo "
    <a href='admin_page.php?{$backparams}'>&lt; — View previous
       " . $this->rowsPerPage . " rows</a>";
?>
    <a href='admin_page.php?<?= $params ?>'>View next <?= $this->
       rowsPerPage ?> rows —&gt;</a>
  </div>
  <div style='padding: 10px;' width='100%'>
    <input type='submit' value='Commit changes'>
    <input type='button' value='Reset' onclick='window.location.
       reload();return false;'>
  </div>
<?
  }

  // Form validation. Returns an array of messages that will be
     displayed to the
  // user. If the array contains zero elements, validation is
     assumed to have
  // succeeded.  This method SHOULD NOT modify $_GET, $_POST,
     $_SESSION.
  function validate($get, $post) {
    return array();
  }

  // Handle form submissions. This method can assume that the
     validation method
  // has succeeded.
  function submit($get, $post) {
    if (!strcmp($post['modified'], ","))
      return;
    $table = $post['table'];
    $modified = explode(",", $post['modified']);
    $modified = array_slice($modified, 1, count($modified)-2);
    $keys = explode(",", $post['keys']);
    foreach ($modified as $row) {
      $set = array();
      $where = array();
      foreach ($keys as $key) {
        $set[] = "‘" . $key . "‘='" . mysql_real_escape_string(
           stripslashes($post["table_{$row}_{$key}"])) . "'";
        $where[] = "‘". $key . "‘='" . mysql_real_escape_string(
           stripslashes($post["table_{$row}_{$key}_previous"])) .
            "'";
      }
      $sql = "UPDATE `{$table}` SET " . implode(", ", $set) . "
         WHERE " . implode(" AND ", $where);
      $_SESSION['messages'][] = $sql;
      mysql_query($sql, $this->connection)
        or die("Error: {$sql}<br>" . mysql_error());
    }
    $myclass = $this->getPageInfo('id');
    header("location: admin_page.php?page={$myclass}&table={
      $table}");
    exit;
  }

  function renderCell($rowindex, $key, $value) {
    $value = htmlentities($value, ENT_QUOTES);
    $valstring = $value;
    if (strlen(trim($valstring)) == 0)
      $valstring = "&nbsp;";
    $output = "<td class='cell' id='table_{$rowindex}_{$key}'
       onclick=\"doEdit('{$rowindex}','{$key}');return false;\"
       title='{$value}'>";
    $output .= $valstring;
    $output .= "<input type='hidden' name='table_{$rowindex}_{
       $key}_previous' value='{$value}'>";
    $output .= "<input type='hidden' name='table_{$rowindex}_{
       $key}' value='{$value}'></td>";
    return $output;
  }

  function join($ary) {
    $str = "";
    foreach ($ary as $key => $val)
      $str .= "&" . $key . "=" . $val;
    return substr($str, 1);
  }
}
```

Listing A.5: Administrative page framework: admin_page.php.

```
/*  admin_page.php
  Created 04/25/05 by birenroy

  Part of the framework used to display a surveyadmin console
  page. */

include_once("../class_utils.php");

include_class("Settings");
include_class("QuestionConfig");
include_class("surveyadmin.NavTree");

session_start();

include("checklogin.php");
if (empty($_GET['page'])) {
  header("location: login.php");
  exit;
}

$pageclass = $_GET['page'];
include_class($pageclass);
$classparts = explode(".", $pageclass);
$classname = array_pop($classparts);
$page = new $classname($_SESSION['org']);
$page->initialize($_SESSION['navtree']);

if (!$page->checkDisplay($_GET, $_POST, $_SESSION)) {
  header("location: admin_page.php?page=" . $page->getPageInfo('
     failed_display_check'));
  exit;
}

if (!empty($_GET['referrer'])) {
  $_SESSION['navtree']->handle_refer($_GET['referrer'], $page->
     getPageInfo('id'));
}
$returnlink = $_SESSION['navtree']->returnlink($page->getPageInfo
   ('id'));
$messages = isset($_SESSION['messages']) ? $_SESSION['messages']
   : array();
unset($_SESSION['messages']);
?>
<html>
```

```
<head>
  <title>Project Assessment</title>
  <link rel='stylesheet' type='text/css' href="../<?= $_SESSION['
     settings']->getURL('stylesheet') ?>">
  <link rel='stylesheet' type='text/css' href="./admin.css">
  <meta http-equiv="Pragma" content="no-cache">
  <meta http-equiv="Expires" content="-1">
  <script type='text/javascript'>
  <!-- // Protect old browsers
  <?= $page->getJavascript() ?>
  // -->
  </script>
  <style type='text/css'>
  <?= $page->getCSS() ?>
  </style>
</head>
<body>
  <form name='mainform' action='admin_page_submit.php' method='
     POST'>
  <input type='hidden' name='page' value="<?= $page->getPageInfo
     ('id') ?>">
  <div class='main'>
  <?= $page->showHeader($_SESSION["settings"]->getURL("
     headerimage"), $returnlink) ?>
  <?= $page->showContent($_GET, $_POST, $messages) ?>
  <?= $page->showFooter() ?>
  </div>
  </form>
</body>
</html>
```

Listing A.6: Administrative page framework: admin_page_submit.php.

```
/*  admin_page_submit.php
  Created 04/25/05 by birenroy

  Part of the framework used to update a surveyadmin console
  page. */

include_once("../class_utils.php");

include_class("Settings");
include_class("QuestionConfig");
include_class("surveyadmin.NavTree");

session_start();

include("checklogin.php");
if (empty($_POST['page'])) {
  header("location: login.php");
  exit;
}

$pageclass = $_POST['page'];
include_class($pageclass);
$classparts = explode(".", $pageclass);
$classname = array_pop($classparts);
$page = new $classname($_SESSION['org']);
$page->initialize($_SESSION['navtree']);
$messages = $page->validate($_GET, $_POST);
if (count($messages) == 0)
  $page->submit($_GET, $_POST);
else
  $_SESSION['messages'] = $messages;
header("location: admin_page.php?page=" . $page->getPageInfo('id'
    ));
exit;
```

Bibliography

[1] Dan Allen. Studs mvc framework+, 2005. http://mojavelinux.com/projects/studs/.

[2] Apache Software Foundation. *The Struts User's Guide*, 2005. http://struts.apache.org/userGuide/.

[3] Bakken, Stig Sæther. *Introduction to PHP.* Zend Technologies, 2000. http://www.zend.com/zend/art/intro.php.

[4] Frederick P. Brooks, Jr. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley, Boston, 1995 edition, 1995.

[5] Steve Burbeck. Applications programming in smalltalk-80a: How to use model-view-controller (mvc). Unpublished, 1992.

[6] Jonathan Cummings. Leading groups from a distance: How to mitigate consequences of geographic dispersion. In S. Weisband & L. Atwater, editor, *Leadership at a distance.* Lawrence Erlbaum Publishers, Mahwah, 2004.

[7] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Boston, 1995.

[8] IBM. *Web design guidelines.* http://www-3.ibm.com/ibm/easy/eou_ext.nsf/publish/572.

[9] Alan Knight and Naci Dai. Objects and the web. *IEEE Software*, March/April 2002.

[10] Fumiaki Shiraishi. System for the online assessment of distributed projects. Master's thesis, Massachusetts Institute of Technology, 2004.

[11] Sun Microsystems, Inc. *Java BluePrints: Model-View-Controller*, 2002. `http://java.sun.com/blueprints/patterns/MVC-detailed.html`.

[12] World Wide Web Consortium. *Cascading Style Sheets, level 1*, 1999. `http://www.w3.org/TR/CSS1`.

[13] World Wide Web Consortium. *HTML 4.01 Specification*, 1999. http://www.w3.org/TR/html4/.

[14] World Wide Web Consortium. *Cascading Style Sheets, level 2 revision 1: CSS 2.1 Specification*, 2004. `http://www.w3.org/TR/CSS21/`.