**CS673F13 Software Engineering** 

**Group Project 2 - MySurvey**

**Project Proposal and Planning**

|  |  |  |  |
| --- | --- | --- | --- |
| Team Member | Role(s) | Signature | Date |
| Fliden | Project Leader | *FD* | 09/24/2013 |
| Gloria | Design Lead | *GJ* | 09/25/2013 |
| Imaad | Implementation lead (joint) | *MK* | 09/26/2013 |
| Chao-Te | Environment and  Integration Leader | *CY* | 09/26/2013 |
| Chih-Chi | Implementation lead (joint) | *CH* | 09/26/2013 |
| Fyodor Wolf | Backup Project Leader / Configuration Leader | *FW* | 09/28/2013 |
| Samuel Tan | Testing Lead | *ST* | 09/28/2013 |

**Revision history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Change** |
| **1.1** | **Fliden Demiri** | **09/30/2013** | **Added Risk Management** |
| **1.2** | **Beth Haines** | **10/01/13** | **Added Non-Functional Requirements** |
| **1.3** | **Gloria Jensen** | **10/06/2013** | **Updated Related Work based on feedback** |
| **1.4** | **Beth Haines** | **10/9/2013** | **Added minimum requirements and clarified wording of “Overview”** |
| **1.5** | **Fliden Demiri** | **10/09/2013** | **Updated metrics section** |
| **1.6** | **Samuel Tan** | **10/12/2013** | **QA Plan update** |
| **1.7** | **Fyodor Wolf** | **10/20/2013** | **Conf. Plan update** |
| **1.8** | **Beth Haines** | **11/7/2013** | **Updated requirements** |

[Overview](#h.87t9hln2vjz0)

[Related Work](#h.mps353x5ezyl)

[Detailed Description](#h.fg3z0hpd4q9v)

[Management Plan](#h.ds8oyr75pnh1)

[Process Model](#h.27177f40uci)

[Risk Management](#h.a4oqwntk3mw)

[Monitoring and Controlling Mechanism](#h.ywdoc2clc9yt)

[Schedule and deadline](#h.tadq5mb0pici)

[Quality Assurance Plan](#h.72e1f4uawy2r)

[Metrics](#h.b2haznn3yyz2)

[Standard](#h.vc72k6dweldv)

[Inspection/Review Process](#h.f1c69ifi68h7)

[Testing](#h.r5d5mhtlf0kq)

[Defect Management](#h.54a4wuncjg1c)

[Process improvement process](#h.jhct37ebxxpn)

[Configuration Management Plan](#h.hw41vg4ykxen)

[Configuration items and tools](#h.bwlb4d4vdox2)

[code commit guidelines](#h.yyauft6zr9hw)

[References](#h.8mva2050iy7t)

[Glossary](#h.ty3i2nqffhtc)

# Overview

This is a semester long project for the BU MET CS473/CS673 course. Our goal is to create an internet application that will provide the ability to create, take and analyze surveys. We anticipate that our survey creators will be data analysts from corporations and from the education sector, that will be looking for data from their designated target group of survey-takers.

The application will also provide pre-fabricated tools to speed up survey creation and publication.

# Related Work

There are many survey tools on the market, some free, like SurveyMonkey, (<https://www.surveymonkey.com/>) which allow the user to login, create the survey, and view analytic reports, and some that are costly but more sophisticated, like Visimojo (<http://visimojo.com/>).

SurveyMonkey will let you create a survey for free and has a question database and templates to choose from. However, more sophisticated features, like choosing question sequencing, require you to subscribe at a higher rate.

Visimojo will let you access the application for free but the application is only in beta mode. Once it is officially released, it will cost a survey-creator $24/month to create surveys.

Our survey tool will be different than those already on the market by offering a range of features geared towards both new survey creators and advanced users. This includes question templates to quickly get started in addition to customizable complex branching, time measurement for each question, and fine-tuned analytic reporting. Survey Monkey and Visimojo each offer many of the features we’re planning in our application, but not all.

# Detailed Description **Functional Requirements** There are two types of users, survey-creators and survey-takers.

# Survey-creators will login and should be able to create a survey, edit it, delete it, view statistics, and get reports.

# Survey-takers will connect through an email link and take the survey. The system will collect the results for reporting.

# Login is pretty straightforward. A survey-creator will register for the site by providing a user ID, password, first name, last name and email address. The system will store this data in the database and use a hashing algorithm to store the password in the database. To login, a survey-creator will enter his login name and password, which the system will verify in the database.

After successfully registering, a survey-creator will be able to edit his account settings,

and change his email address or password.

A survey-taker will be able to connect through his email link. A cookie will be placed on the survey-taker’s machine the first time he completes the survey. If the cookie already exists on the survey-taker’s machine, he will be denied access to the survey.

# To create the survey, the system will ask for a survey name, and then create a unique URL for that survey. The survey-creator will be presented with question choices, such as "multiple choice", "multiple answer", "true or false", "ranking", "single choice", "short answer with keywords", "Comments". The survey-creator will enter the question and the possible answer choices. There could also be questions that have no answer choices, such as "Comments" type questions.

The application will also provide some template questions that the survey-creator can

select. The survey-creator can then incorporate the template questions into the survey, or modify the existing question to make it unique for his survey.

# As part of the question entry, the survey-creator will also enter a flag to determine whether or not an answer is required, and any required error-checking to determine if the questions are outside a given range. The survey-creator can also set a time quantum for the response so that, for instance, a survey-taker can only take 1 minute to answer one particular question, although for another question the survey-taker might be allowed 5 minutes, or indefinitely for his response.

# When the survey is complete, the survey-creator reviews the questions, and links answer choices to the next question in sequence. For instance, if the question is something like "Are you a student at BU?" the answer choice "Yes" might link to a question like "What's your favorite class?", but if the answer choice is "No", the next question might be "Are you a faculty member"? The question itself will also have a default link to the next question in case no answer is required.

# When the sequencing is complete, the survey-creator will have an option to preview the survey before submitting it. The survey-creator will be asked to confirm that the survey should be created.

A survey-creator can also edit a survey template or a survey he’s previously created and have the option to keep it as is, or save it under another name. When editing, the survey-creator will have the option to add question-answer sets, to modify the text of existing questions and answers, to modify the number of answers, and to change the sequence links for any given answer. He will also be able to delete any question-answer set.

At any point while creating or editing the survey, the survey-creator can save the survey and come back to it at a later date.

The survey-creator will be able to publish the survey when it’s complete and un-publish it in case he finds an error. He will also be able to set a time when the survey should be published.

The survey-creator will also have the option to delete any survey that he has previously created as long as there have been no responses to his survey.

When the survey is complete, the survey-creator will log out of the application.

The survey-creator will be responsible for supplying the survey link to the survey-takers. Survey-takers will connect to the website with the URL in their email. The system will collect their answers to make the data available for analytic reports.

When the survey-taker finishes the survey, he will be presented with a “Thank-You” screen to let him know the survey is finished, a cookie will also be stored on the survey-taker’s system to prevent the same survey-taker from retaking the survey again.

Once any number of survey-takers have taken the survey, the survey-creator will be able to view the answers that have been input on their survey, as well as some statistical reports about the survey.

**Non-Functional Requirements**

1. The system should be available 98% of the time.

Survey-creators will become discouraged if they are unable to access the site and may take their business to another competitor. Survey-takers may answer the survey when they get a free moment.

So if they are unable to access the site, it becomes more likely that they will abandon the survey and not return. If this happens, the results for the survey will not be complete. For these reasons, the system will be up and ready 98 percent of the time.

2. The system will aim for response time of less than half a second.

If survey-creators have to wait too long for a response, they will become discouraged and take their business elsewhere. If survey-takers have to wait too long for a response, they also will become discouraged and give up. Both kinds of users expect and deserve very fast response time. We will make sure that our coding is up to industry standards and optimized for performance. We will also replicate the data onto a backup server that can replace the "live" server if performance is slow.

3. The system will not collect any personal data from the survey-takers.

Privacy has become a major concern with any online activity. The survey-takers must feel free to submit their answers as anonymously as possible, or they will hesitate to take a survey. Because of this, the system will not retain any information from the survey-taker other than explicit answers to survey questions.

4. All system data will be backed up nightly to non-volatile data.

Data is our business. If the data from any survey is lost, it could be difficult or impossible to recreate it. For that reason, all data will be backed up nightly to disk. In addition, the database will be synchronously replicated with the database on the backup server.

5. Data for surveys that have ended for more than 180 days will be archived to another data source.

Performance will become an issue if the size of the database grows too large. All data for surveys that ended 180 days before the current date will be saved off in an archive database and purged from the live database.

6. The system will implement industry-standard security measures.

Password security for the survey-creators will be handled with a hashing algorithm. Measures will be taken to prevent SQL injection. Site security will depend on survey-takers having the correct link. If either the survey-creators or survey-takers feel that the site is not secure, they will not want to submit the survey questions or survey answers.

7. The system will operate using any of the major browsers, such as Google Chrome, Internet Explorer, Mozilla Firefox and Opera.

For the convenience of the survey-takers, the application will function equally on all major browsers. Having this flexibility will make it easier for the survey-takers to submit their responses, which will make it more likely for them to respond.

**Minimum Set of Requirements**

The minimum functionality for this project will include:

1. User Login

A survey-creator will be able to login and gain access to his surveys by entering his username and password.

2. User Registration

A new user will be able to create an account with username, password, first and last names.

3. Create Survey

A survey creator will be able to create a new survey with a unique URL and will be able to add basic questions to the survey that could have different question types, such as single choice (choose the one correct answer), multiple answer (choose all of the correct answers for that question) and a question for entering comments.

4. View Created Surveys

A survey creator will be able to view all the surveys and survey questions that he has entered onto the site.

5. Take Survey

A survey-taker will be able to access the survey and to enter his answers.

# Management Plan

## Process Model

This project will follow a non-strict Agile development process, focused on features and collaboration. There will be three iterations (excluding the initial planning phase), three weeks each. The initial phase will be focused on planning, organization and defining requirements. The other three iterations will be more typical Agile iterations. Each iteration will include:

· Planning and requirement analysis

· Design, implementation and testing

· Update and add documentation

As per the Agile standards, each iteration will focus on features. After each iteration there will be a demo demonstrating the features completed during the iteration. Also there should be a working software after each iteration (although with minimal features). The features developed during each iteration will be decided during the initial phase but are also subject to change along the process (as per the Agile standards). There will be a close collaboration between the client and the project team. The client and the team together will decide the project requirements and the features completed during each iteration. This collaboration will continue during the entire process.

## Objectives and Priorities

The quality of the software and the client satisfaction will be the main objective. We will give the quality of code and the overall quality of the software the highest priority. Feature quality will also have priority over the quantity of features developed. Considering the type of project and the strict, fixed, limited time to complete it, schedule will also be a high priority.

## Risk Management

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Risk** | **L** | **I** | **R** | **P** | **Retirement or mitigation plan** | **Strategy** | **Responsible team member** |
| 1 | Feature under estimation | 4 | 10 | 1 | **7** | Keep each feature simple. Make them independent. Use Pivotal Tracking to keep track and help estimate. Re-estimate after each iteration. | Avoid | Beth |
| 2 | Inexperience with PHP | 5 | 10 | 4 | **24** | Read and practice with PHP | Conqueror | Fliden, Fyodor |
| 3 | Inexperience with YII framework | 8 | 6 | 2 | **30** | Read and practice with YII framework | Conqueror | Fyodor |
| 4 | Lack of communication | 1 | 7 | 1 | **40** | Weekly meeting and continuous communication | Avoid | Fliden, Imaad |
| 5 | Inexperience with Git and the feature branch workflow | 9 | 4 | 3 | **42** | Read and practice with Git | Conqueror | Fliden, Fyodor |
| 6 | Time shortage | 7 | 8 | 4 | **48** | Start from a minimum viable product. Re-estimate features and tasks after each iteration. Use Pivotal Tracker to help with task management and estimation. | Avoid | Chao-Te |
| 7 | Team member absence | 1 | 5 | 1 | **60** | Continuous communications and collaboration. Always have a back up plan. | Conqueror | Gloria |
| 8 | Requirements ambiguity | 3 | 7 | 2 | **64** | Continuous communications and collaboration. Elaborate requirements before and after each iteration. | Avoid | Beth |
| 9 | Imbalance in workload | 5 | 5 | 2 | **72** | Decide task assignment together. Keep track of weekly report. | Avoid | Chih-Chi Hsu |

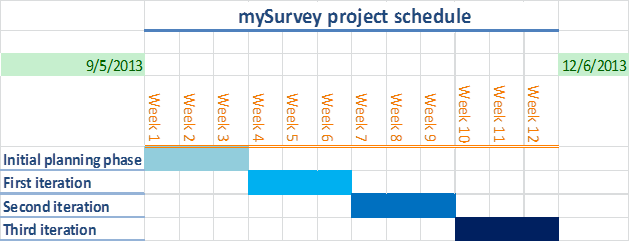
***L*** *- Likelihood (1-10),* ***I*** *- Impact (1-10),* ***R*** *- Retirement cost (1-10)*

***P*** *- Priority (11-Likelihood)\* (11-Impact)\*Retirement Cost*

## Monitoring and Controlling Mechanism

There will be a meeting every week after class to discuss what has been done in the previous week and plan the next week. Every team member will also complete a weekly report with tasks completed during the week and the corresponding time spent completing them. This information will help estimate features for each iteration. In addition, after each group meeting, one of the team members (in a rotating matter) will complete a short report (minutes report) about the meeting. This will help track of what is discussed during the meeting.

## Schedule and deadlines



1. The first iteration starts 09/26/2013 and ends 10/17/2013. It will focus on designing the database and creating the basic screens, like login and CRUD operations (creating/updating/deleting) for questions and answers.
2. The second iteration starts 10/17/2013 and ends 11/07/2013. It will be the polishing, like providing more question types like templates (or generic questions about age, gender, etc. at a minimum) and more survey creation features.
3. The third iteration starts 11/07/2013 and ends 12/06/2013. It will bring in the survey-takers piece, and maybe include some basic reports. It may also add additional survey creation features like sequencing answers to questions.

# Quality Assurance Plan

## Metrics

**Product size:**

* **Lines of code:** We will keep count of the lines of code. We will use git to create the statistics. We will run the following commands to get the statistics for each team member:

git log --author="<authorname>" --oneline –shortstat

git log --author="<authorname>" --pretty=tformat: --numstat

After each iteration, we will run this to keep track of the group’s workload and to help estimating the next iteration.

**Product quality /reliability:**

* **Unit Tests:** We will keep track of all the unit tests and run a report after each iteration with the number of unit test per user story. All features should pass all written unit tests in order to assure their proper functionality. Generally most methods should have at least one unit test. Every feature (branch in Git) should have at least one unit test.
* **Errors:** The MySQL, Apache and PHP error logs will be used to log exceptions and warnings. We will keep track of the number of errors and warnings after each iteration.
* **Issues and bugs:** Github will be used for tracking any issues and bugs with the code. Its pull request system will be used to apply patches to fix issues and bugs. The number of bugs, the number of bugs related to each user stories and the number of open vs closed bugs will be tracked after each iteration. Average times to fix a bug will also be tracked.

**Process and group efficiency:**

* **Work hours:** Our weekly report will be used to keep track of the hours spent by each team member every week and the respective tasks. After each iteration we will run a report to see compare our effort to the other team members and to help estimate the next iterations.

## Standard

Coding standard: We will base our coding standard on the Yii php framework standards [Conventions][Code Style]. Functions and variables names are in camel case, which capitalizes the first letter of each word in the name and joins them without spaces. The first word should be all in lowercase. Class names on the other hand, will also follow camel case convention but their first word should also be capitalized to differentiate from variables and functions (e.g. $averageNumber, sendResponse(), SurveyQuestion). In addition, private class member names will have an underscore character (e.g. $\_stateValue).

Class files and view files will be named after their corresponding public class and view name. Classes will also be prefixed with a [TBD] letter to prevent conflict with other integrated classes.

Each method and class should also have an explanatory comment. @param, @var, @property and @return together with the corresponding type should be used in every comment [Code Style].

## Inspection/Review Process

## Documentation review:

Subjects: MOM, WSR(Weekly Status Report), SRD, Test Cases,

Defect Report, Release Notes

When: During the implementation of each iteration

Who: Entire team

How: Peer review is recommended at offline while group.

Discussion can be conducted periodically after class.

All the review defects will be put into Github issue tracking

system.

## Code review:

Subjects: Source code

When: During the implementation of each iteration

Who: Entire team

How: Peer review is recommended at offline while group.

Discussion can be conducted periodically after class.

All the review defects will be put into Github issue tracking

system.

## Testing - Unit Test

· Who: Each developer

· When: During the implementation phase of each iteration

· How to track: Code Review (peer/ group review if required)

PHPUnit (or other unit test tool that makes team comfortable can be applied - needs to be decided during next team meeting)

Defect tracking system

Defect report

**- Integration Test**

· Who: Selected testers (Entire Team)

· When: End of iteration 2, iteration 3

· How to track: Manual test

Defect tracking system

Defect report

**- System Test**

· Who: Selected testers (Entire Team)

· When: During iteration 3

· How to track: Manual test

Defect tracking system

Defect report

**- User Acceptance Test**

· Who: End customer (Professor or other groups)

· When: After iteration 3

· How to track: Manual test

Defect tracking system

Defect report

## Defect Management

We will use github’s internal issue tracking system for the defect management of this project.

Defect is defined as any issue that will cause the application behavior unexpectedly, it may include (but not limited to) the following:

**Critical** (Priority High):

- The defect affects critical functionality or critical data, such as application crash, no response, critical data loss, etc.

**Major** (Priority High):

- The defect affects major functionality or major data. It has a workaround but is not obvious and is difficult.

**Minor** (Priority Medium):

- The defect affects minor functionality or non-critical data. It has an easy workaround. Example: A minor feature that is not functional in one module but the same task is easily doable from another module.

**Trivial** (Priority Low):

- The defect does not affect functionality or data. It does not even need a workaround. It does not impact productivity or efficiency. It is merely an inconvenience. Example: UI layout discrepancies, spelling errors.

The entire team is responsible for the defect management while the QA leader can play a role to take the lead.

# Configuration Management Plan

## Configuration items and tools

Configuration management will be responsible for the configuration of the following items:

* Programming environment (PHP, Apache, MySQL) and the IDE (Netbeans/Eclipse/Coda)
* Version control software (Git) and repository (Github)

For the duration of the project. Developers will have the choice of choosing the best Integrated Development Environment (IDE) to suit their needs. Like the IDE, developers are also allowed to choose the proper development stack (a pre-configured PHP, MySQL, Apache installation) that best suits their needs. In lieu of providing flexibility for the coder to choose the tools that most effect for him or her, no code that is specific to a developer’s IDE or stack configuration is to be committed to the project’s code repository. Any and all code that is committed to the repository, should work on every team members IDE and stack.

In order to achieve a rigid code base that will work on all developer’s machines, the developer’s stack configuration must support the following features:

* PHP 5.4
* defining virtual hosts (Apache)
* htaccess rewrite rules (Apache)
* disabling php short tags (Apache)
* creating databases (MySQL)
* defining database usernames and passwords. (MySQL)

A developer’s stack will need to be configured to support the strictly defined settings of the MySurvey project. These stack features will provide the necessary configurable flexibility for each developer’s stack, while allowing the code to remain free of user specific configurations.

## **Change management and branch management**

The project will follow the feature branch development pattern where each feature requires itself to have it’s own branch. For the purposes of this document, the master branch will refer to the remote origin master branch that resides on Github. The feature branch will refer to the branch that a user is currently working on. Once the programmer has completed the feature he/she will push the branch to the repository. Once the feature has been reviewed and approved it will be merged to the master branch.

The master branch will serve as the most up to date version of the completed and deployable codebase. Any features that are incomplete or not tested, should remain in their own respective branch until they have been completed and tested for any bugs. Once the code has been completed and tested with the newly implemented features, one must first *pull* the code from the master branch and re-run all of the project tests that have been added to master until that point. This will ensure that only working code is published to the master branch. Only once all tests are completed, can the feature branch be *pushed* to master.

## Code commit guidelines

Any branch that is actively being worked on, should be published to the github repository. This is to inform everyone of what’s being worked on, and what dependencies exist within the code.

Code must first be pulled from remote and tested before being pushed to remote branches. This will prevent others from branching off of your buggy code.

Any feature branch that is being pushed to remote, must first be merged (pushed) into it’s parent branch first. This will encourage proper feature ancestry and facilitate the parent’s branch merging down the line.

Committing changes to the framework folder of this project is prohibited. Yii framework is maintained by a whole community of developers and if you really wish to modify the framework, only consider doing so in the [yii official repo](https://github.com/yiisoft/yii). The Framework folder for this project should contain only an official release of the yii framework.

# References

"*Code Style*" *YII Framework/Github*. N.p., n.d. Web. 23 Sept. 2013.

<https://github.com/yiisoft/yii/wiki/Core-framework-code-style>

"*Conventions.*" *YII Framework*. N.p., n.d. Web. 23 Sept. 2013.

<http://www.yiiframework.com/doc/guide/1.1/en/basics.convention>

# Glossary