Software Design Document for:

Kirkland Signature Online Survey Tool

Prepared by:
Josh Pfeffer, Jeremy Koletar, Joe Thomas, Patrick Cook, Tim Wong

Version 1.0

November 18, 2016

Table of Contents

1.0 Introduction

This section provides an overview of the entire requirement document. This document describes all data, functional and behavioral requirements for software.

1.1 Goals and objectives

In this document, we will be going over what our survey tool system will look like tangibly as well as what will be done to create the system. There will be diagrams and pictures to show mock ups of what the UI for our system will look like. This document will primarily describe the implementation of the Kirkland Survey Tool described in our previous vision and scope document.

1.2 Statement of scope

The survey tool will consist of three major functions. These functions include the implementation of creating a survey, answering a survey, and analyzing survey data. All of these functions are essential to the survey and will be focused on throughout the process of creating this survey tool.

Regarding creating a survey, XYZ Corp. employees will be able to create targeted surveys through our web based software. They will be able to manage email lists containing potential voters and send targeted emails to those voters. Users will also be able to utilize the common types of survey questions such as multiple choice and ranking questions, but will also be able to use conjoint analysis trade-off questions.

Survey respondents will be able to answer the survey by means of an email link. Their responses will be stored in a data bank as they answer. The respondents can only answer the survey once.

XYZ Corp. analysts will be able to analyze the survey data as soon as respondents begin to answer questions. They will be able to view the data in forms of graphs and charts. They will also be able to look at the data through cross-tabulation, regression analysis, and conjoint analysis tables.

1.3 Software context

The Kirkland Signature Online Survey Tool analytics portion will be used internally by the XYZ Corporation. The server will be hosted in-house, and the survey data will be available exclusively for internal company use. XYZ Corporation will use this data to advise various political campaigns. Survey takers will be able to access the site via a link

that they will receive in an email. Survey creators and analysts will access the site with a username and password via an intranet URL.

1.4 Major constraints

- The quality of our product is largely dependent on the hardware specifications (e.g. server quality, storage capacity, processing speeds, etc.) that XYZ Corp decides to implement for this project.
- A secure infrastructure to ensure that sensitive, private information does not get leaked.

2.0 Data design

2.1 Internal software data structure

The Kirkland Signature Online Survey Tool's internal data structure will be split into two parts: client-side (the web browser) and server-side (the web server).

On the client side, surveys, survey responses, and various analytics will be stored locally within the web browser until later saved or submitted.Classes will be accessed via a Model-View-Controller system. Information will be permanently stored in a MySQL database.

The server-side and client-side will communicate via HTTP requests. Some data used for client-side rendering will be passed as JSON objects.

2.2 Global data structure

The main global data structure being used for this application will be a MySQL database. The database structure will contain all data required to represent users, surveys, survey responses, and email lists. The web application will communicate and send HTTP requests to the web server, which will then communicate directly with our MySQL database to create, read, and update data.

2.3 Temporary data structure

Temporary data structures used in this application will mostly be limited to view-specific data sent to the client for display in the form of JSON objects. Temporary data is ephemeral and thus not stored; generated once, transported from the web server to the web browser, and the data is then inserted into the browser's current web page.

2.4 Database description

```
-- Create syntax for TABLE 'branch_condition'
CREATE TABLE `branch_condition` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `pivot_question_id` int(11) unsigned NOT NULL,
  `destination_question_id` int(11) unsigned,
  `sort_number` int(11) unsigned DEFAULT NULL,
  `next_condition_id` int(11) unsigned DEFAULT NULL,
  `test_question_id` int(11) unsigned NOT NULL,
  `conditional` varchar(1) DEFAULT NULL,
  `not` tinyint(1) DEFAULT NULL,
```

```sql
  `test_value` text,
  PRIMARY KEY (`id`),
  FOREIGN KEY (`pivot_question_id`) REFERENCES `question`(`id`),
  FOREIGN KEY (`destination_question_id`) REFERENCES `question`(`id`),
  FOREIGN KEY (`test_question_id`) REFERENCES `question`(`id`),
  FOREIGN KEY (`next_condition_id`) REFERENCES `branch_condition`(`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Create syntax for TABLE 'employee'
CREATE TABLE `employee` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `username` varchar(255) DEFAULT NULL,
  `password` varchar(255) DEFAULT NULL,
  `name` varchar(255) DEFAULT NULL,
  `email` varchar(255) DEFAULT NULL,
  `reset_hash` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Create syntax for TABLE 'mailing_list'
CREATE TABLE `mailing_list` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `name` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Create syntax for TABLE 'question_response'
CREATE TABLE `question_response` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `question_id` int(11) unsigned NOT NULL,
  `respondent_id` int(11) unsigned NOT NULL,
  `response_value` text NOT NULL,
  PRIMARY KEY (`id`),
  FOREIGN KEY (`question_id`) REFERENCES `question`(`id`),
  FOREIGN KEY (`respondent_id`) REFERENCES `respondent`(`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Create syntax for TABLE 'respondent'
CREATE TABLE `respondent` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `email` varchar(255) NOT NULL DEFAULT '',
  `name` varchar(255) NOT NULL DEFAULT '',
  `mailing_list_id` int(11) unsigned NOT NULL,
  PRIMARY KEY (`id`),
  FOREIGN KEY (`mailing_list_id`) REFERENCES `mailing_list`(`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Create syntax for TABLE 'respondent_metadata'
CREATE TABLE `respondent_metadata` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `respondent_id` int(11) unsigned NOT NULL,
  `key` varchar(255) NOT NULL,
  `value` varchar(255) NOT NULL,
  PRIMARY KEY (`id`),
```

```sql
  FOREIGN KEY (`respondent_id`) REFERENCES `respondent`(`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Create syntax for TABLE 'survey'
CREATE TABLE `survey` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL DEFAULT '',
  `mailing_list_id` int(11) unsigned NOT NULL,
  `editable` tinyint(1) NOT NULL DEFAULT '1',
  `nonce` varchar(64) NOT NULL,
  PRIMARY KEY (`id`),
  FOREIGN KEY (`mailing_list_id`) REFERENCES `mailing_list`(`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Create syntax for TABLE 'question'
CREATE TABLE `question` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `survey_id` int(11) NOT NULL,
  `name` varchar(255) NOT NULL DEFAULT '',
  `question_type` varchar(255) NOT NULL DEFAULT '',
  `serialized_form` blob,
  PRIMARY KEY (`id`),
  FOREIGN KEY (`survey_id`) REFERENCES `survey`(`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

3.0 Architectural & component-level design

3.1 System Structure

The system as a whole will function via the interaction of the following classes: Respondent, MailingList, Survey, IQuestion, QuestionResponse, Employee, QuestionBank, BranchCondition, and Mail. Some of these classes will also interact with the MySQL database to save, retrieve, and edit information.

3.2 Respondent Class

The Respondent class represents an individual who has responded to a survey.

3.2.1 Processing narrative (PSPEC)

This is a simple data container class. Respondents are instantiated by the MailingList class, and tracked in the respondent list of the MailingList instance that created it. The Respondent class has the following attributes:
- Email - the respondent's email address, to which survey invitations will be sent.
- Name - the respondent's name.
- Metadata - arbitrary additional information about the respondent. May be used for survey targeting.

3.2.2 Interface description.

Respondent
  Respondent(email: String, name: String, metadata: Map<String, String>)
  getEmail(): String
  getName(): String
  getMetaData(key: String): String
  getMetaDataKeys(): Set<String>
  setEmail(String)
  setName()
  setMetaData(key: String, value: String)

3.2.3 Processing detail

3.2.3.1 Design Class hierarchy

Respondent does not extend any class or have any child classes. Instances of Respondent belong to one or more MailingList instances.

3.2.3.2 Restrictions/limitations

The Respondent class does not have any restrictions or limitations.

3.2.3.3 Performance issues

Metadata is represented as a Hashtable, which can grow arbitrarily large. If a very large amount of metadata is stored, space usage will be very large. Lookup and write operations maintain an amortized fixed time. Large amounts of metadata are not anticipated.

3.2.3.4 Design constraints

The Respondent class does not have any design constraints.

3.2.3.5 Processing detail for each operation

Respondent(email: String, name: String, metadata: Hashtable<String, String>)
> Simple constructor, sets all fields.

getEmail(): String
> Returns the email field.

getName(): String
> Returns the name field.

getMetaData(key: String): String
> Access a value in the metadata Hashtable with the given key. Returns the associated value if it exists, otherwise null.

getMetaDataKeys(): Set<String>
> Returns an iterable Set containing all keys in the metadata Hashtable.

setEmail(String)
> Sets the email field.

setName()
> Sets the name field.

setMetaData(key: String, value: String)
> Add a key-value pair to the metadata Hashtable if the key does not already exist, or replace the value associated with the given key with the specified value if the key already exists.

3.3 MailingList Class

The MailingList class represents a list of potential survey respondents.

3.3.1 Processing narrative (PSPEC)

When an XYZ Corp employee wishes to create a new mailing list, they pass in a name, and a new MailingList class is instantiated. A variety of methods are provided to allow adding respondents existing respondents to the list, creating new respondents and adding them, or removing respondents. The MailingList class has the following attributes:

- Name - the name associated with the lists
- Respondents - a list of all respondents contained in the list.

3.3.2 Interface description.

MailingList

      MailingList(name: String)
      getRespondents(): List<Respondent>
      addRespondent(respondent: Respondent)
      addRespondent(email: String, name: String, metadata: Map<String, String>)
      addRespondents(respondents: List<Respondent>)
      addRespondents(contactCSV: CSVFile)
      removeRespondent(respondent: Respondent)

3.3.3 Processing detail

3.3.3.1 Design Class hierarchy

MailingList does not extend any class or have any child classes. Instances of MailingList hold 0 or more Respondent instances.

3.3.3.2 Restrictions/limitations

The MailingList class is dependent on org.apache.commons.csv.CSVParser for parsing respondent information from CSV files.

3.3.3.3 Performance issues

Run time and memory usage scales linearly to size when loading respondents form a CVS file. For very large files, this could cause a noticeable delay, but this matches user expectations when importing data from large files. There are no performance considerations for the other methods of this class.

3.3.3.4 Design constraints

MailingList names should be unique, or XYZ Corp employees may have difficulty distinguishing different instances.

3.3.3.5 Processing detail for each operation

      MailingList(name: String)

Constructs a new instance of MailingList, and sets the name field.

addRespondent(respondent: Respondent)

Appends the given respondent to the list referenced by the respondents field.

addRespondent(email: String, name: String, metadata: Map<String, String>)

Constructs a new instance of Respondent with the specified arguments, and passes it to addRespondent(Respondent).

addRespondents(respondents: List<Respondent>)

Append the specified respondents to the end of the list referenced by the respondents field.

addRespondents(contactCSV: CSVFile)

Retrieves the header row from contactCSV, and the creates a new Respondent for each subsequent row, matching the 'email' and 'name' arguments of the Respondent constructor to the column with the same header in contactCSV. The Respondents are stored in a List, and once all rows of contactCSV have been parsed, the resultant List is passed to addRespondents(List<Respondent>)

removeRespondent(respondent: Respondent)

The list of respondents associated with the MailingList is searched for the specified respondent, and if found, it is removed from the list.

3.4 Survey Class

The Survey class represents the actual survey that will be created and used. Each class has its own survey ID and questions (instances of the IQuestion class). It will use the MailingList class to send out the survey after the survey is completed.

3.4.1 Processing narrative (PSPEC) for Survey class

When a user builds a survey, a new Survey object is created. The following information corresponds with the survey:
- Survey name
- Number of questions
- IQuestion objects
- Nonce

3.4.2 Survey class interface description.

Survey

createSurvey(name : String) : Survey
addQuestion() : boolean
removeQuestion(ndx : int) : boolean
addQuestion(question : IQuestion, ndx : int)
getQuestion(ndx : int) : IQuestion
generateAccessKey(respondent: Respondent): String

isValidKey(key: String, email: String): boolean

3.4.3 Survey processing detail

The Survey methods defined above are relatively straightforward. There will be no complex algorithms used in these methods as the methods are simple. Each survey will be having an arraylist of questions and the methods will work with the arraylist.

3.4.3.1 Design Class hierarchy for Survey

The Survey class has no parent or child classes. However, each instance of Survey has an arraylist of associated IQuestion objects.

3.4.3.2 Restrictions/limitations for Survey

There are no restrictions or limitations currently to be implemented in the Survey class.

3.4.3.3 Performance issues for Survey

Since the Survey object is to be used and accessed through a web browser, there should be no performance issues for this class. The only potential performance issue for this class would be if the web browser was unable to retrieve the Survey object IQuestions from the database, which would be highly unlikely.

3.4.3.4 Design constraints for Survey

Because this class relies on the IQuestion class, if the IQuestion class method for some reason fails, the Survey object would also fail. In addition, this class allows only one question to be created and added at a time.

3.4.3.5 Processing detail for each operation of Survey

createSurvey(name : String) : Survey
- This method takes in a String containing the name of the survey. It will create a Survey object with the String representing the unique object, generate a random nonce for controlling access to the survey, and return the Survey object.

addQuestion()
- This method adds an IQuestion object to the last available position in the arraylist containing all the IQuestion objects. This method will return true if the IQuestion was successfully added to the arraylist.

removeQuestion(ndx : int) : boolean
- This method takes in the index (question number) of the question that is desired to be removed. Using the arraylist remove method, it will remove the IQuestion object at the desired index. This method will return true if the IQuestion was successfully removed from the arraylist.

addQuestion(question : IQuestion, ndx : int) : boolean
This method takes in an IQuestion object and the index of where the object is desired to be placed in. It then adds an IQuestion object to the index position in

the arraylist containing all the IQuestion objects. This method will return true if the IQuestion was successfully added to the arraylist.

getQuestion(ndx : int) : IQuestion
This method takes in an index and returns the IQuestion object at the index in the arraylist.

generateAccessKey(respondent: Respondent): String
This method takes a Respondent, accesses the respondent email field, appends it to the Survey's private nonce, and returns a secure hash (sha256) of the result.

isValidKey(key: String, email: String): boolean
This method appends the given email to the survey's private nonce, hashes the result using the same method used by generateAccessKey, and compares the result to the key given as an argument. Returns true if the values are equal, otherwise false.

3.5 IQuestion Class

The IQuestion class is meant to be a general representation of a question that can be created within a survey, and its components are: name, type (multiple choice, trade-off, or free response), and the actual choices/data.

3.5.1 Processing narrative (PSPEC)

When a user creates a question from within the survey, an IQuestion object is instantiated. The following fields belong to a IQuestion:
- Name
- Type (multiple choice, trade-off, or free response)
- Choices / data (stored in JSON format)

The instantiated IQuestion object is called upon every time question data is needed to be accessed during execution. For example, when a user creates a new question, a new IQuestion object is instantiated and the user assigns a name, a question type, and choices/data to said object.

3.5.2 Interface description

IQuestion:
getQuestionType() : int
addBranchCondition(destination: IQuestion, condition: String, testVal: int, currentQuestion: IQuestion) : boolean
getResponses() : JSON

3.5.3 Processing detail

This class is used for data storage and retrieval, so there are no algorithms implemented within the code. getQuestionType() and getResponses() are accessors, while addBranchCondition() is a setter.

3.5.3.1 Design class hierarchy

The IQuestion class has neither parent nor child classes. However, a Survey object contains a variety of IQuestion objects. A QuestionResponse object corresponds to a particular IQuestion.

3.5.3.2 Restrictions/limitations

There are no realistic or anticipated restrictions.

3.5.3.3 Performance issues

We do not anticipate any performance issues arising with the implementation and runtime of this class. Two of the associated methods are accessors, and the other method is a setter, which will not lead to much overhead. However, communication with the server and database may be impaired in certain, unusual circumstances such as network outages and other related issues.

3.5.3.4 Design constraints

The only major design constraint is if the addBranchCondition() method returns false, which indicates that the method call failed. This would leave the intended branch condition in limbo.

3.5.3.5 Processing detail for each operation
- getQuestionType() : int
  - Operating as an accessor, this method returns an enumerated type of the question in the form of an integer.
- addBranchCondition(destination: IQuestion, condition: String, testVal: int, currentQuestion: IQuestion) : boolean
  - This method creates a new branching condition using the destination question, current question, test value, and condition for which to test against.
- getResponses() : JSON
  - Operating as an accessor, this method returns a JSON data structure with the various responses to said IQuestion.

3.6 QuestionResponse Class

The purpose of the QuestionResponse class is to represent a user's response to numerous types of questions using a simple class that can be used in many different scenarios. The QuestionResponse class contains the following components: a question, survey, respondent and response value.

3.6.1 Processing narrative (PSPEC)

When a user answers a question from within a survey, a QuestionResponse object is created. A QuestionResponse contains the following attributes:
- Survey - used to specify the survey that contained the question being answered
- Question - used to specify the question that the response is associated with
- Respondent - used to keep track of the respondent answering the question
- Response value - use to record the answer to the question, stored as an int array.

3.6.2 Class description.

QuestionResponse:
- getResponseValue() : int [ ]

3.6.3 Component Processing Detail

A QuestionResponse object is responsible for storing a user's response during runtime. Due to the simplistic nature of our class design, no algorithms will be implemented within our code.

3.6.3.1 Design Class hierarchy

The QuestionResponse class has neither parent nor child classes, instead it is used as a temporary structure for representing a user's response. Each QuestionResponse corresponds to a specific IQuestion. This allows the system to retrieve all responses for a specific IQuestion.

3.6.3.2 Restrictions/limitations

We do not expect any restrictions or limitations unless more questions types are added. We have designed this class to represent all supported IQuestion types. However, if addition IQuestion types are added our QuestionResponse class might not be able to represent user responses to questions of that type.

3.6.3.3 Performance issues

We do not anticipate any performance issues. Our implementation stores responses in a simple format, an int array. This format will help reduce data usage/storage and will allow the system to use generic methods instead of question-specific methods for processing data for use in data analysis.

3.6.3.4 Design constraints

A major design constraint comes from the representation of a question's response. We are storing the response in an int array for all question types except free response questions. This design heavily simplified other sections of our system, however, the addition of new question types may lead to design changes.

3.6.3.5 Processing detail for each operation
- getResponseValue() : int [ ]
    - Operating as an accessor, this method returns an integer array which is capable of representing all but one type of question response. Multiple choice, tradeoff, rating, ranking, and constant sum questions can all be represented using an integer array. This design choice will add code reusability and increase readability.

3.7 Employee Class

The Employee class is intended to be a representation of a Kirkland Signature Online Survey Tool admin user (a non-survey-taker). The following fields will be used to represent an Employee from within the application: name, username, password, email, and a reset hash.

3.7.1 Processing narrative (PSPEC)

Whenever an employee creates an account within our system, an Employee object is created. The following fields are stored within this class:
- Name
- Username
- Password
- Email
- Reset hash

This Employee object is used to uniquely identify an employee within the company.

3.7.2 Interface description

Employee
checkCredentials(username: String, password: String) : boolean

3.7.3 Processing detail

Because the Employee class is solely used for data retrieval and validation, there are no algorithms implemented within this class. The class will only use accessors to communicate with the server and database.

3.7.3.1 Design Class hierarchy
The Employee class has neither parent nor child classes.

3.7.3.2 Restrictions/limitations
There are no practical restrictions to this class that we anticipate at this time.

3.7.3.3 Performance issues

In general, we do not anticipate any major performance issues that will arise in relation to the Employee class. There will be no processor-intensive calculations or algorithms being made. However, performance issues or errors may arise in the event of a network outage or hiccup of some sort. In this case, communication to and from the server and database could be interrupted.

3.7.3.4 Design constraints

The only obvious design constraint is if an invalid username and/or password is passed into the checkCredentials() function, in which the function would return false.

3.7.3.5 Processing detail for each operation

- checkCredentials(username: String, password: String) : boolean
  - This method essentially validates an employee's credentials (username and password). It will do so by communicating with the database. If the said credentials exist within the database, the function will return true. If not, the function will return false.

3.8 QuestionBank Class

The QuestionBank class allows us to store question templates which can then be added to any future surveys. The QuestionBank will act like an accessor and will not use a local data structure to store questions. Instead, the QuestionBank class will be used communicate with the database to store, load and remove question templates.

3.8.1 Processing narrative (PSPEC)

When a user creates a survey they will be presented the option to **load** a question template from our QuestionBank or create a new question. If a user creates a question they will have the option to **save** the question as a template for later use. The user will also be able to view a list of question templates that have been saved and they will have the option to **remove** a question template from the database.

3.8.2 Class Description

QuestionBank:
saveQuestion(question : IQuestion) : boolean
loadQuestion(questionID : int) : IQuestion
removeQuestion(questionID : int) : boolean

3.8.3 Component Processing Detail

The QuestionBank is used only for accessing data and will not require any algorithms. Instead, this class will communicate with the database to load, save, and remove question templates from the database.

3.8.3.1 Design Class hierarchy

The QuestionBank class has neither parent nor child classes. Instead, it acts as a standalone class and is used for database communication. The QuestionBank class will work closely with the IQuestion class to access information specific to the question template being saved, loaded, or removed.

3.8.3.2 Restrictions/limitations

There are no anticipated restrictions or limitations.

3.8.3.3 Performance issues

The QuestionBank class will communicate directly with the database. The performance will be reliant on the health of the connection to the database and the internet speed/reliability.

3.8.3.4 Design constraints

The QuestionBank class has been limited to the following functions: saveQuestion(), loadQuestion(), removeQuestion(). These functions will provide all the required functionality for our system, however, other functions may be added in the future depending on their usability, scalability and integration with the rest of the system.

3.8.3.5 Processing detail for each operation

- saveQuestion(question : IQuestion) : boolean
  - In order to save the question template this function will create a new IQuestion object which is saved to the Question table in our database. The IQuestion's survey_id will be set to the value "-1" to represent that it does not belong to a specific survey, instead, it represents a question template which can be used in future surveys.
- loadQuestion(questionID : int) : IQuestion
  - This method uses a questionID, represented as an integer, to pull a question template from the database. The question will be duplicated and added as a new row to the Question database table. The survey_id will then be set to the survey currently being created. This allows the question template to keep the survey_id of "-1", representing a template question versus a survey question.
- removeQuestion(questionID : int) : boolean
  - This method uses a questionID to remove the question template, with the corresponding questionID, from the database.

3.9 BranchCondition Class

The BranchCondition class is a hybrid utility class and storage class. It contains sufficient processing logic and data to determine whether or not a particular question should branch or not at a pivot point.

If respondent over 30 and male, branch to question "n"

```
            id=1                          id=2
     pivot_question_id=4          pivot_question_id=4
 destination_question_id=null  destination_question_id=99
        sort_number=1               sort_number=null
     next_condition_id=2         next_condition_id=null
      test_question_id=1          test_question_id=2
        conditional='>'             conditional='='
           not=0                        not=0
       test_value=30              test_value='Male'
```
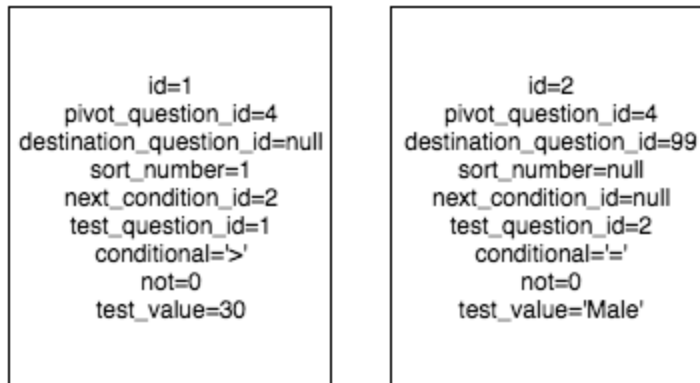
3.9.1 Processing narrative (PSPEC)

When adding or removing branching logic from a question, one or more **BranchConditions** will be created. Each BranchCondition represents part or whole of a boolean expression, which is to be evaluated against the respondent's responses so far.

3.9.2 Interface description

BranchCondition

public static eval(questionID : Integer, respondentID : Integer) : Integer | null
private eval(respondentID : Integer) : Integer | null

3.9.3 Processing detail

3.9.3.1 Design Class hierarchy

The BranchCondition class has no parent classes, implements no interfaces, and has no subclasses.

3.9.3.2 Restrictions/limitations

The BranchCondition's eval behavior at runtime is extensively based on the validity of its data. Bad or malformed data in the database could cause infinite recursion, or other resource exhaustion.

3.9.3.3 Performance issues

BranchCondition's is primarily constrained by the complexity of conditionals constructed by the survey creator.

3.9.3.4 Design constraints

None.

3.9.3.5 Processing detail for each operation

- public static eval(questionID : Integer, respondentID : Integer)
  - When given the question and respondent IDs through its public eval method, the BranchCondition class first selects all BranchConditions that use the specified question as a pivot point. It then builds a local map of BranchCondition IDs to BranchCondition instances. Next, it builds a list of BranchConditions which have a non-null sort_number, sorted by this value. The static method then iterates through list or BranchConditions and calls the private "eval" method on the instance. If any of the BranchConditions evaluated return a non-null result, this result is returned to the caller of the static method. If the BranchCondition returns a null result, the next BranchCondition loaded will be evaluated until there remain no more BranchConditions to be evaluated, and null will be returned to the caller of the static method.

- private eval(respondentID : Integer)
  - The private, non-static "eval" method will query for the appropriate QuestionResponse from the database for the respondent's response to the test_question. Once it has the response value from the test_question, the instance will then perform the comparison specified in its instance variables.
  - If the result of the evaluated boolean expression is false, the instance will return null, to indicate a failure to branch.
  - If the result of the evaluated boolean expression is true, the instance will do one of two actions: if the instance has a "next_condition" designated, it will load that particular condition from the database, and return the result of the next_condition's eval to its caller. If the instance has no next_condition designated, it will return its destination_question_id.

3.10 Mail Class

The Mail class is a utility class which handles the sending of emails.

3.10.1 Processing narrative (PSPEC)

When a survey is finished and ready for respondents, a MailingList is passed to this class to dispatch invitation emails to all respondents in the MailingList.

When an XYZ Corp employee initiates a password reset, the Employee object represting their account is passed to this class to send them a password reset link.

The Mail class has the following static fields defined in the class:
- Server - the hostname of the mail server.
- Port - the port of the SMTP server running on the specified host.
- From - the email address used for the From field.
- Username - username for authenticating with the SMTP server.
- Password - password for authenticating with the SMTP server.

3.10.2 Interface description.

Mail

mailToEmployee(employee: Employee, subject: String, body: String)
mailToList(list: MailingList, subject: String, body: String, survey: Survey)

3.10.3 Processing detail

The Mail class does not involve and complex processing.

3.10.3.1 Design Class hierarchy

The Mail class does not extend any classes or have and child classes.

3.10.3.2 Restrictions/limitations

Emails sent by the Mail class are text only. Future versions may implement the sending of email with an HTML body.

3.10.3.3 Performance issues

The sending of mail may be delayed based on connection quality to the SMTP server. Sending of mail may fail, which will require retrying.

3.10.3.4 Design constraints

3.10.3.5 Processing detail for each operation
- mailToEmployee(employee: Employee, subject: String, body: String)
  - Accesses the employee email from the Employee object
  - Connects to the SMTP server, authenticates
  - Sends an email to the employee with the specified subject and body
- mailToList(list: MailingList, subject: String, body: String, survey: Survey)
  - Accesses the list of respondents from list
  - Accesses the email of each respondent
  - Connects to the SMTP server, authenticates
  - For each respondent, generates a survey access link consisting of:

- - - Survey id
    - Respondent Email
    - Survey Access Key, obtained via Survey.generateAccessKey()
  - Sends an email to each respondent, with the specified subject, the the specified body with the generated survey access link appended.

4.0 User interface design

A description of the user interface design of the Kirkland Signature Online Survey Tool will be presented in this section.

4.1 Description of the user interface

A detailed description of user interface including screen images or prototype is presented.

4.1.1 Objects and actions

The next several pages outline and describe the following user interfaces/menus:

For Survey Creators:
- Login Page
- View surveys
- Create Survey
- Adding a multiple choice question to a survey
- Adding a free response question to a survey
- Adding a rating question to a survey
- Adding a constant sum question to a survey
- Adding a ranking question to a survey
- Adding a conjoint analysis question (two choice) to a survey
- Adding a conjoint analysis question (rating)

For Survey Respondents:
- Answer a multiple choice question
- Answer a free response question
- Answer a rating question
- Answer a constant sum question
- Answer a ranking question
- Answer a conjoint analysis

For Survey Analysts:
- Login Page
- View Statistics
- View in Graph/Chart
- View Trade-Off Analysis
- View Cross Tabulation
- View Regression

User Interfaces for Survey Creators

Login Page:

- Survey creators and analysts will be presented with a login screen which will then direct them to a dashboard where they can begin designing or analyzing surveys.

Create Survey:

Welcome Admin Home

Create New Survey

View Surveys

Manage Email Groups

New Survey

Survey Name: Name

Start Building Survey

- After a survey creator logs in they will have the option to create a new survey. This process begins by entering a name which is followed by clicking the "Start Building Survey" button.

Adding a multiple choice question to a survey:

Welcome Admin

Home	Survey1

Question #1 | + Add Question

QUESTION BANK

🔍 Search for question

Demographics ▶

Education ▶

Political Views ▶

Events ▶

Multiple Choice ▼   Single Answer ▼

Question: [   ]

Pick the most favorable option

◉ Option 1

+ Add Set Option   + Add User Response

- To add a multiple choice question to the survey the survey creator can press the "Add Question" button and select from the drop down menu the specific type of question they would like to add.
- A question outline is shown and the creator has the following options:
  - Set question title
  - Add options
  - Add user response

Adding a free response question to a survey:

Welcome Admin

Home	Survey1

| Question #1 | + Add Question |

QUESTION BANK

🔍 Search for question

Free Response ▼

Demographics ▶

Education ▶

Political Views ▶

Events ▶

Question: [   ]

Write your own response

Response...

- To add a free response question to the survey the survey creator can press the "Add Question" button and select from the drop down menu the specific type of question they would like to add. The survey creator then inputs the question title.

Adding a rating question to a survey:

Welcome Admin

Home	Survey1

Question #1	+ Add Question

QUESTION BANK

🔍 Search for question

Demographics ▶

Education ▶

Political Views ▶

Events ▶

Rating ▼   Rating Scale [5 ▲▼]

Question: [   ]

Rate the following question

Strongly Disagree   Strongly Agree

1   2   3   4   5

○   ○   ○   ○   ○

- To add a rating question to the survey the survey creator can press the "Add Question" button and select from the drop down menu the specific type of question they would like to add. The survey creator can then add a question title.

Adding a constant sum question to a survey:

- To add a rating question to the survey the survey creator can press the "Add Question" button and select from the drop down menu the specific type of question they would like to add. The survey creator can then add a question title, add the options and set the "Total Points to be Allocated" field to their desired number.

Adding a ranking question to a survey:

Welcome Admin

Home	Survey1

Question #1 | + Add Question

QUESTION BANK

🔍 Search for question

Demographics ▶

Education ▶

Political Views ▶

Events ▶

Ranking ▼   Range for Ranking: 4 ⬍

Question: [_____]

Rank the following options from most favorable to least favorable in ascending order

1 ⬍ Option 1

2 ⬍ Option 2

3 ⬍ Option 3

4 ⬍ Option 4

- To add a ranking question to the survey the survey creator can press the "Add Question" button and select from the drop down menu the specific type of question they would like to add. The survey creator can then add a question title, select the number of options used for the ranking question, and then modify the options.

Adding a conjoint analysis question to a survey:

Welcome Admin

Home	Survey1

Question #1 | + Add Question

Conjoint Analysis ▼   Rating Question ▼   Rating Scale [5 ▲▼]

QUESTION BANK

🔍 Search for question

Demographics ▶

Education ▶

Political Views ▶

Events ▶

Question: [   ]

Rank how preferable you are to this option

Option 1

[Attribute 1 ▼] [Level 1 ▼]

[Attribute 2 ▼] [Level 1 ▼]

+Add Attribute

Not Preferred   Preferred
1   2   3   4   5
○   ○   ○   ○   ○

- To add a rating question to the survey the survey creator can press the "Add Question" button and select from the drop down menu the specific type of question they would like to add. The survey creator then will select the number of choices for the question, add a question title, add attributes to each option, and set the levels for each attribute.

Welcome Admin

Home | Survey1

Question #1 | + Add Question

Conjoint Analysis ▼ | Two Choice ▼

QUESTION BANK

🔍 Search for question

Demographics ▶

Education ▶

Political Views ▶

Events ▶

Question: [   ]
Pick the most favorable option

Option 1

| Attribute 1 ▼ | Level 1 ▼ |
| Attribute 2 ▼ | Level 1 ▼ |

+Add Attribute

Option 2

| Attribute 1 ▼ | Level 1 ▼ |
| Attribute 2 ▼ | Level 1 ▼ |

+Add Attribute

- To add a rating question to the survey the survey creator can press the "Add Question" button and select from the drop down menu the specific type of question they would like to add. The survey creator can then create the question, rename the options, and select the attributes and levels associated with each attribute. The survey creator can add as many attributes as desired.

User Interfaces for Survey Respondents

The following section covers each question interface. The user is presented with a question which they must complete before proceeding. To reduce redundancy in the following section we have chosen to include descriptions only for non-trivial questions.

Answer a multiple choice question:

Survey Name

Question 1:

1. What political party do you align with the most?

- ⊙ Democratic
- ○ Republican
- ○ [Other]

[Next]

Answer a free response question:

Survey Name

Question 2:

2. Please enter comments about the outcome of the most recent election:

[Comments...]

[Next]

Answer a rating question:

Survey Name

Question 3:

4. Rate the degree to which you align with your political party's ideologies.

1	2	3	4	5
○	○	○	○	○

Strongly Disagree   Strongly Agree

Next

Answer a constant sum question:

Survey Name

Question 4:

4. Please distribute 100 points between the following options in order of importance.

[30] Housing

[40] Education

[20] Food Accessibility

[10] Financial Assistance

Next

- The user must distribute 100 points between multiple options. Our backend will verify the input and confirm whether the points add up to the desired amount.

Answer a ranking question:

Survey Name

Question 5:

6. Rank each option from lowest prefered to highest preferred

- [2 ⬍] Housing
- [1 ⬍] Education
- [3 ⬍] Food Accessibility
- [4 ⬍] Financial Assistance

Next

- The user must rank each option without reusing the same ranking number. Our backend will verify the user's response in order to ensure each option is ranked uniquely.

Answer a conjoint analysis question:

Survey Name

Question 6:

6. Choose the most favorable option

Candidate 1

Attribute 1: level 1

Attribute 2: level 2

Attribute 3: level 3

◯

Candidate 2

Attribute 1: level 1

Attribute 2: level 2

Attribute 3: level 3

◯

Next

Answer a checkbox question:

Survey Name

Question 7:

7. Select all that apply

☑ You align with a specific political party

☑ You are registered to vote

☐ You voted on propositions this year

Next

View Surveys:

Welcome Admin

Home

Create New Survey

View Surveys

Manage Email Groups

All Surveys

▼ Survey Title	▼ Published	▼ Response Rate	▼ View/Edit
Obama Approval Nov 7	Yes	75%	Go
Obama Approval Nov 14	No	N/A	Go

- A survey creator has the option to view basic information about each survey. They will have the option to click a specific survey and edit them by pressing the "Go" button.

User Interfaces for Survey Analysts

Login: Please reference the login UI in our "User Interfaces for Survey Creators" section.

View Statistics:
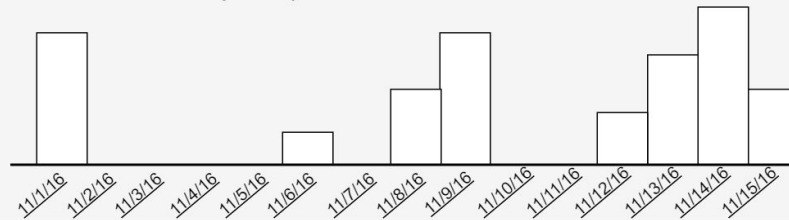
- The analysts can access the analysis side after clicking the "view/edit" link from the "view survey" tab in the home page. They will be directed initially to the statistics page which will have the number of total responses to the survey, the number of surveys sent, the percent responded, and the scale of responses since the survey was sent out.

View in Graph:

Welcome Admin

Home	Survey1 Analysis
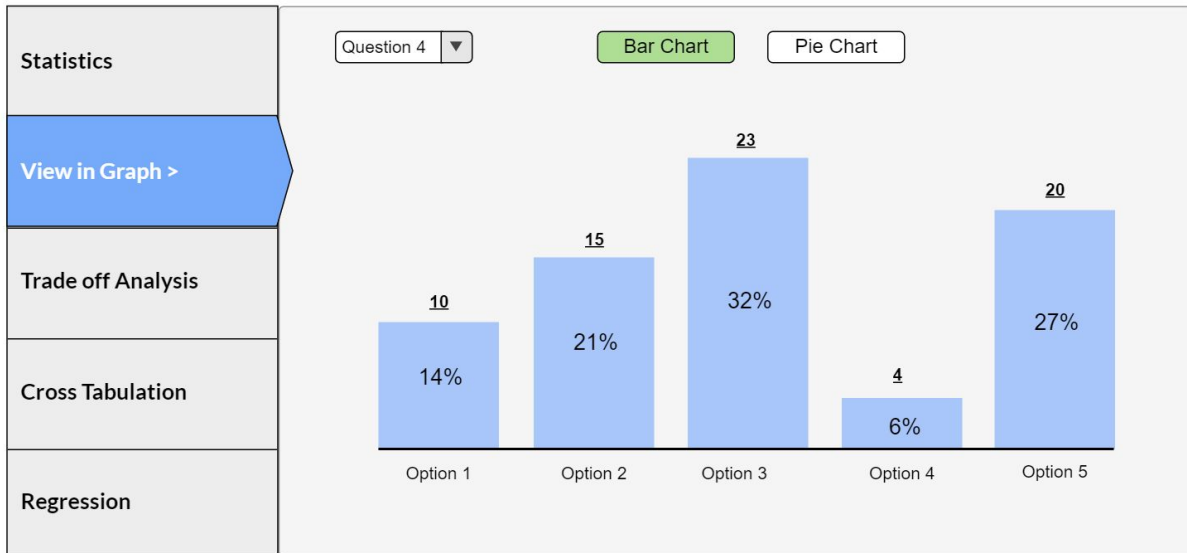
Statistics	Question 4 ▼ Bar Chart Pie Chart
View in Graph >	
Trade off Analysis	
Cross Tabulation	
Regression	

- The analysts can access the analysis side after clicking the "view/edit" link from the "view survey" tab in the home page. They can then click the "view in graph" option where they can pick the question they want to analyze. After that, they can pick either to view the data in a bar chart or a pie chart. Clicking either option will bring up a respective chart with the number and percentage of responses for each option.

View Trade-Off Analysis (Cont.):

Welcome Admin

| Home | Survey1 Analysis |

Statistics

View in Graph

Trade off Analysis >

Cross Tabulation

Regression

Question 9 ▼

| ▼ Card 1	▼ Card 2
Attribute 1: Level 1	Attribute 1: Level 3
Attribute 2: Level 2	Attribute 2: Level 1
Attribute 3: Level 1	Attribute 3: Level 3
Attribute 4: Level 3	Attribute 4: Level 2
Attribute 5: Level 3	Attribute 5: Level 3

Total Responses: 27   33

- The analysts can access the analysis side after clicking the "view/edit" link from the "view survey" tab in the home page. They can then click the "Trade off Analysis" option where they can pick the question they want to analyze. Only conjoint analysis questions will avaliable to pick from. If the question is a two choice question, the system will show both cards (options) along with the attributes and levels associated. At the bottom, the total number of responses for each card will be shown.

View Trade-Off Analysis (Cont.):

Welcome Admin

Home	Survey1 Analysis

Statistics	Question 9 ▼

Ranking out of **5**

	▼ Card	▼ Attribute 1 level	▼ Attribute 2 level	▼ Preference

Statistics				
	1	1	1	5
View in Graph	2	1	2	0
	3	1	3	0
Trade off Analysis >	4	2	1	1
	5	2	2	4
	6	2	3	5
Cross Tabulation	7	3	1	3
	8	3	2	3
Regression	9	3	3	2

- The analysts can access the analysis side after clicking the "view/edit" link from the "view survey" tab in the home page. They can then click the "Trade off Analysis" option where they can pick the question they want to analyze. Only conjoint analysis questions will be avaliable to pick from. If the question is a rating question, the system will display all the cards that have been associated with the card in question whether or not they have been asked in the survey. On the right hand column, the average preference for the card will be shown.

View Cross Tabulation Analysis:

- The analysts can access the analysis side after clicking the "view/edit" link from the "view survey" tab in the home page. They can then click the "Cross Tabulation" option where they can pick question 1 and question 2. The options from question 1 will be listed on the left column and the options from question 2 will be listed on the top row. The boxes in between will show the percentage of respondents who answered the respective options. The third question in the bottom will be an optional question that can be tested against each box in the table.

View Regression Analysis:

Welcome Admin

| Home | Survey1 Analysis |

Statistics

View in Graph

Trade off Analysis

Cross Tabulation

Regression >

Question 1:

| Question 5 | ▼ |

Attribute 1

15

10

5

-20 -10   10   20   30   40   50   60   Attribute 2
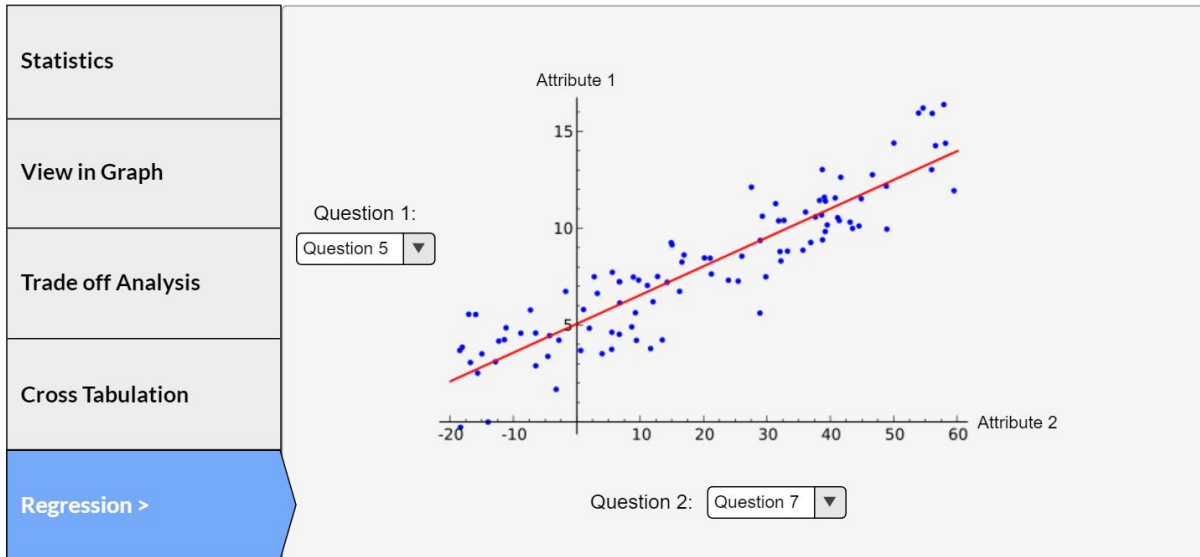
Question 2: | Question 7 | ▼ |

-   The analysts can access the analysis side after clicking the "view/edit" link from the "view survey" tab in the home page. They can then click the "Regression" option where they can pick the questions they want to analyze. The system will then graph the responses and plot a best fit line to correspond to the answers. (regression graph taken from Wikipedia)

4.2 Interface design rules

Our interface design rules for the Kirkland Signature Online Survey Tool are taken from Ben Shneiderman's "Eight Golden Rules of Interface Design". Below, we list a description of each rule as well as how it applies to our survey tool:

1. **Strive for consistency.**
   a. Consistent sequences of actions should be required in similar situations; identical terminology should be used in prompts, menus, and help screens; and consistent commands should be employed throughout.
   b. The color scheme is consistent throughout the survey tool (blue, grey, and white with green buttons). Survey creation, survey viewing, and email management all live in the same menu, making for ease of use. Adding questions to a survey is similar in terms of UI, regardless of question type. The same goes for responding to a survey.
2. **Enable frequent users to use shortcuts.**
   a. As the frequency of use increases, so do the user's desires to reduce the number of interactions and to increase the pace of interaction. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.
   b. Regular users of the system must make a username and password, allowing for easy system access each and every time. You can also manage question banks and mailing lists for later reuse.
3. **Offer informative feedback.**
   a. For every operator action, there should be some system feedback. For frequent and minor actions, the response can be modest, while for infrequent and major actions, the response should be more substantial.
   b. For each and every type of submission via the click of a button, the user will be subsequently informed with a fading confirmation box, detailing which action they just completed.
4. **Design dialog to yield closure.**
   a. Sequences of actions should be organized into groups with a beginning, middle, and end. The informative feedback at the completion of a group of actions gives the operators the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans and options from their minds, and an indication that the way is clear to prepare for the next group of actions.
   b. When a survey has been completed and submitted, a question has been successfully added to the bank, or a mailing list has been created, the user will be informed of such--that the process was completed successfully.
5. **Offer simple error handling.**
   a. As much as possible, design the system so the user cannot make a serious error. If an error is made, the system should be able to detect the error and offer simple, comprehensible mechanisms for handling the error.

b. If a user fills out a field incorrectly or incompletely when they press the submit button, they will be given suggestions to help fix their mistake.

6. **Permit easy reversal of actions.**
   a. This feature relieves anxiety, since the user knows that errors can be undone; it thus encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data entry, or a complete group of actions.
   b. Most user actions will permit easy reversal. However, as of now, we will not allow survey respondents to go backwards to a question that they have already answered.

7. **Support internal locus of control.**
   a. Experienced operators strongly desire the sense that they are in charge of the system and that the system responds to their actions. Design the system to make users the initiators of actions rather than the responders.
   b. Users with a username and password will be able to create new surveys, view surveys, manage email groups, and add questions to the question bank.

8. **Reduce short-term memory load.**
   a. The limitation of human information processing in short-term memory requires that displays be kept simple, multiple page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics, and sequences of actions.
   b. Our system will be simplistic and extremely user-friendly, in which each page will be designed around completing a singular task (e.g. view survey, answer a question, or create a new survey)

4.3 Components available

Bootstrap 4 provides many built-in components that can be used for layout design, functional design and user interaction. The following HTML components will be used extensively:
   ● Divs, inputs, buttons, tables, links and scripts

4.4 UIDS description

● The user interface will be developed using the Bootstrap 4 framework.
● The Bootstrap 4 framework will be used to build all user interfaces and provide a "skeleton" for our backend code to work with.
● Our backend code will be written in Java and will interact with the interfaces created. This encompasses database communication and front-end data population.

5.0 Restrictions, limitations, and constraints

One limitation that we have is the access and use of internet. As we have mentioned before in our SRS document, this whole system is going to be web-based. If the users and respondents do not have internet access or a working web-browser, they will not be able to use our system to access and/or answer surveys.

Another constraint we have is that only the question types provided and the formats provided will be able to be used. There is no way to create a custom question that does not follow the formats provided. If additional question types or formats are desired, we are willing to look into implementing them in the future, but currently, no other question types will be allowed into the survey.

A limitation we have currently is that the data received is not going to be easily exported into an excel document or the like. While the diagrams can be captured as an image and data can be copied manually, we do not provide an option to export.

The final limitation that is imposed on the respondents is that they have to have an email and be able to access it. Our survey tool depends on the use of emails to send surveys to respondents so if they don't have one, they cannot answer the survey. In addition, there are no other ways of sending a survey except by email.

6.0 Appendices

Presents information that supplements the design specification.

6.1 Requirements traceability matrix

	Respondent	MailingList	Survey	IQuestion	QuestionResponse	Employee	QuestionBank	BranchCondition	Mail
UC-1						X			
UC-2			X						
UC-3			X	X					
UC-4			X	X			X		
UC-5			X	X			X		
UC-6			X	X					
UC-7			X	X			X	X	
UC-8			X						
UC-9	X		X	X	X				
UC-10			X	X	X				
UC-11			X	X	X				
UC-12			X	X	X				
UC-13			X	X	X				
UC-14			X	X	X				
UC-15			X	X	X				

6.2 Packaging and installation issues

Software will be delivered as a JAR package. The deployment server should have Java Enterprise Edition 7 installed.

6.3 Design metrics to be used

We will closely track the following design metrics throughout the use of our product: complexity, documentation, bugs, maintainability, code quality, reliability, security, and tests.