CHALMERS

Refining the workflow of a web survey management application

A Bachelor Thesis in Computer Science and Engineering

FILIP CARLÉN
HAMPUS FORSVALL
SIMON PERSSON
ERIK THOLÉN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2015

Refining the workflow of a web survey management application
Filip Carlén,
Hampus Forsvall,
Simon Persson,
Erik Tholén,

Examinator: Dag Wedelin

Bachelor Thesis: 2015:40
Department of Computer Science and Engineering
Chalmers University of Technology
412 96 Gothenburg, Sweden
Phone: +46 (0)31-772 1000

Abstract
This bachelor thesis describes the process of improving the workflow for marketing analysts creating online questionnaires by supplementing the currently used legacy software with a new web application. The goal is to reduce time spent on creating questionnaires in order to allow for more time analyzing the results. Design of graphical user interfaces (GUI) is an area of research in constant change, regularly affected by new trends. This causes existing design to quickly become outdated, sometimes greatly affecting the usability of the applications behind them.

The workflow was improved by first analyzing the existing software and its shortcomings in terms of user experience (UX), and then developing a new application with these in mind. The new application would allow for a better workflow along with new features that the existing software did not have, e.g generating documents compatible with Microsoft® Word and storing commonly used questionnaires in a database for later reuse.

The result is a web application called QuestBuilder that can be accessed through a web browser. The application is tailored to the needs of marketing analysts and offers a handful of new features. The application is considered a prototype and therefore does not cover all features from the old software. However the features that are implemented offer a significantly improved workflow compared to the old software.

Sammanfattning

Detta kandidatarbete beskriver processen av att förbättra arbetsflödet för marknadsanalytiker som skapar onlineformulär genom att komplettera den nuvarande mjukvaran med en ny webbapplikation. Målet är att reducera den tid som spenderas på att skapa frågeformulär så att användarna kan ägna mer tid åt att analysera resultaten istället. Design av grafiska gränssnitt är ett område under ständig förändring, regelbundet påverkat av nya trender. Detta medför att existerande design snabbt blir utdaterad och i vissa fall påverkar användbarheten hos applikationerna nämnvärt.

Arbetsflödet förbättrades genom att först analysera den existerande mjukvaran och dess tillkortakommanden med avseende på användarvänlighet för att sedan utveckla en ny applikation med dessa i åtanke. Den nya applikationen skulle erbjuda ett bättre arbetsflöde med nya funktioner som den existerande mjukvaran saknade, till exempel möjligheten att generera dokument kompatibla med Microsoft® Word och att lagra vanligt förekommande frågeformulär i en databas för senare återanvändning.

Resultatet blev en webbapplikation kallad QuestBuilder som kan bli åtkommen via en webbläsare. Applikationen är skräddarsydd efter marknadsanalytikers behov och tillgodoser en mängd nya funktioner. Applikationen anses vara en prototyp och har därför inte alla funktioner som den gamla mjukvaran har. Däremot erbjuder de funktioner som faktiskt är implementerade ett signifikant förbättrat arbetsflöde.

Acknowledgements
We wish to thank everyone that has helped in the writing and proof-reading of this thesis as well as in the development of the application by providing feedback and technical help. Without the assistance of our supervisor Dag Wedelin the outcome of this project would not have been the same. We express our gratitude to the company we worked towards and their representatives Jakob Werner and Christophoros Antoniades for their continuous support and feedback on the developed application.

Contents

Vocabulary

Agile - Iterative way to run projects.

AMA - American Marketing Analysts. A fictional name for the company the prototype has been developed for.

AngularJS - A popular JavaScript framework.

Back End - In this project a collective word for all code running on the server side of the application.

Cloud Computing - When processing power is accessed through the web instead of being in the personal computer.

Cognitive Walkthrough - A way to evaluate GUI.x

Cryptographic Protocols - Techniques to make data secure.

Data Binding - The synchronisation of data between model and view components.

Desktop application - Software running locally on a computer.

DOM - Document Object Model. Describes the content of a website.

Front End - In this project a collective word for all code running on the client side of the application.

Git A version control software

GUI - Graphical user interface. What the user of a software sees and works with.

HCI - Human-Computer Interaction. The science of how humans understand and work with computers.

Heuristic Evaluation - A way to evaluate GUI.

High-functionality Application - Software most commonly used in the business world. Can be recognized by the amount of features implemented.

HTML - Static markup language used to create web pages.

Java - An object oriented programming language.

Java Virtual Machine (JVM) - The virtual environment in which the Java compiler is run

JavaScript - A scripting language offering some object oriented options.

jQuery - A JavaScript library.

JSON - A open standard format for storing data using key-value pairs

Legacy Software - Outdated software using old design principles.

PERL - A scripting language.

Personas - Make believe persons used for evaluating GUI.

Product owner - The individual responsible for the product in a Scrum project.

QuestBuilder - The prototype web application software built in this project.

Questionnaire - An inquiry sheet.

ROI - Return on investment, A way to measure profit.

Scrum - An agile development method.

UX - User Experience. How pleasant a software is to work with.

Version Control - Used to keep track of software when multiple people work on it.

Waterfall model - A start to end, no change allowed, way of running projects.

Web Application - A website with behaviour similar to a desktop application.

Wire-framing - The act of creating mock-ups of interface before implementation.

Workflow - A collective word for how a certain kind of work is carried out.

WYSIWYG - What-You-See-Is-What-You-get

1. Introduction

A common problem with software tools built for managing complex and large data structures is to design an intuitive user interface for them[Fischer, 2001]. Often it is the case that the end users of such a tool end up trying to adapt their workflow to the poorly designed user interface, instead of the other way around. The software controls the user when in fact, the user should control the software. Thus the benefits of using the software are greatly reduced and the software does not achieve its full potential.

The ideal situation would be where the software does not limit the capabilities of the user but instead empowers them and provides a way for the user to easily realize their potential. Users of software are seldom as technical as the people who created it and therefore learning and using the software can be very time-consuming.

A poor user interface can result in extensive learning curves and an unnecessarily time-consuming workflow. In worst case, the end user might even end up never using some of the more advanced features that the program provides. There is a risk that the software is not utilized to its full potential and the end users spends more time than necessary to get their actual work done. And as time often equals money, this could result in actual profit loss for a company using such software.

This project is about finding user interface shortcomings in an existing software that is used to create web surveys and to produce a new application that improves upon these shortcomings.

1.1. Purpose

The thesis aims at improving the workflow of producing online questionnaires for American Marketing Analysts (AMA), by developing a prototype that provides a more efficient process. This new workflow will be tailored to the specific needs of AMA, simplifying previously repetitive and tedious tasks.

1.2. Limitations

The final product produced by this project is considered to be a prototype. Due to the nature of the legacy application, the prototype will not be a standalone application. Instead the prototype will act as a middleware that aims to enhance a selection of features from the legacy application.

1.2.1. Functionality

The original software application, which the prototype is meant to complement, includes a vast amount of features. The development process has been geared towards implementing a subset of these features in the prototype, in addition to new features in order to improve the workflow at AMA. The reason for this being that AMA has requested a certain subset of features that need to be implemented.

1.2.2. Testing

The purpose of testing within a software engineering project is to ensure that the end product is secure, solid and fulfills the purpose of the application [ISTQB, 2015]. The prototype will continuously receive feedback from user tests at AMA in order to tailor the prototype to their needs. This feedback will be considered as the main testing aspect of the project. In addition, the new application will be manually tested to ensure that it is able to open and save files compatible with the legacy application.

1.2.3. Security

The prototype will be deployed and running at the intranet used by AMA. Access is therefore only granted to employees at AMA or through a VPN (Virtual Private Network) connection to the intranet. Due to this, no additional security measures are considered necessary.

1.3. Background

The American Marketing Analyst (AMA) company is a multinational corporation specialising in the highly competitive field of creating and analyzing industry standard questionnaires. As a part of their survey creation process AMA currently use a legacy software with many features. However, this software is cumbersome and makes the entire process unnecessarily tedious for the employees.

In order to counter the negative effects of the legacy software AMA began an inhouse project aimed towards improving their workflow. Unfortunately the lack of competence in software development at the company hindered the process and an alternative solution was sought. As a result they contacted the Information Technology program at Chalmers University of Technology and proposed this bachelor thesis.

Instead of creating a new desktop application it was decided to develop a complementary web application featuring new ideas and techniques. By deploying the new application in the cloud it would become more accessible and future proof.

1.4. Problem

This section will cover the problem that this thesis aims to solve. In order to improve the workflow of AMA, the problem will be broken down into the following parts:

- Finding shortcomings of the current workflow
- Evaluation of the legacy application
- Developing the new application

These parts will ultimately contribute to a software solution prototype that will improve the workflow of AMA.

1.4.1. Workflow

In order to develop the prototype, it is of high value to find the weaknesses and shortcomings of the current workflow utilized at AMA to identify what features the prototype should include. These features must counteract the weaknesses and shortcomings of the current workflow. In order to gain an understanding of what features are required to do so, discussions with AMA must be held.

1.4.2. Legacy application

The prototype aims not only to improve the general workflow at AMA, but also to offer an enhanced user experience in comparison to the legacy application. The prototype must offer much of the same functionality that is offered in the legacy application. However, the prototype must offer this functionality in a more intuitive and user-friendly manner.
In order to improve the user experience, the legacy application must be evaluated. The results will then be taken into account when developing the new application.

1.4.3. The new application

In order to ensure that the new application fulfills the requirements it must also be evaluated in the same manner as the evaluation of the legacy application. The purpose of the evaluation is to compare the web application with the legacy application.
The new application must also cancel the negative aspects identified in the old workflow. To ensure this, continuous discussion with AMA must be held regarding the new workflow that the new application offers.

1.5. Method

This section describes the most common techniques used to realize this project. These techniques cover project planning and execution. It also covers how the source code has been shared between all collaborators and the programming languages used to develop the prototype.

1.5.1. Agile Development

The chosen software development method was Agile. Agile development is a collection of several software project methodologies that is used to organize software projects resulting in better end products [Dyba, 2008]. The main idea of Agile development focuses upon four pillars [Beck, 2001]. The core notion of these pillars is that time consuming activities, such as documentation and contract negotiations are down prioritized. Functioning software and collaboration is valuable. Also, individuals should interact with each other directly instead of using external tools and processes. The last pillar mentions that changes of plans should not be an issue.

The agile methodology chosen for this project is Scrum. In Scrum, the software is developed by self organizing teams. The team should be cross-functional and consist of a Product Owner, a development team and a Scrum Master. [Schwaber, 2013] The Product Owner is responsible for the product and for maximizing the value of it, while the Scrum Master ensures that the Scrum process is applied without any major friction.

1.5.2. Programming Languages

Early in the process it was decided to use Java in the back end of the application due to the group being familiar with the language, thus easing the process of development. It was also decided to use JavaScript for the logic in the front end. JavaScript differs from Java but is similar enough that the group felt comfortable using it. Since the resulting software would be a web application, HTML was also used in order to create the views.

1.5.3. Software evaluations

Two methods were chosen in order to evaluate the legacy software and the prototype. The first method that was chosen is Heuristic Evaluation, a well recognized and adopted method of evaluating software regarding GUI[Nielsen, 1994]. The second method is Cognitive Walkthrough, a method used to discover the flaws of interface design down to graphical component level.[Spencer, 2000]

1.5.4. Version Control

In order to ease collaboration and push out releases frequently the use of version control was considered a necessity. The method agreed upon was Git. Git eases the process of creating branches in a project to allow for independent work. These branches can then be merged together in order to create a release [Driessen, 2010]. To further ease the use of Git the plugin Git-flow was used. Git-flow promotes the use of master, develop and feature branches, making it simple to track the work done by all members and allowing for a structured development phase. [Kummer]

2. Theory

The specific problem considered in this project is an example of the more general problem of creating software that covers a great deal of functionality. In this chapter we will give an overview of the more general problem and describe the historical development of web-based software and why it could be preferable over desktop applications.

2.1. Migrating desktop applications to the cloud

This section will provide some background on the evolvement of the web in order to motivate why developing a web application is beneficial and in some cases even favored over developing a desktop application.

2.1.1. Background

The world wide web has developed into a network of interconnected computing devices and resources more powerful than most of us could have ever imagined. This network has lead us to yet another large scale revolution in the field of computing and information technology. The era of desktop computing is transitioning into the era of the web. Comparing today's computing with the early days of computing, the web can now, in many ways, be considered to be an infinitely powerful, shared mainframe computer that is accessible by everyone [JOTHIRAMALINGAM, 2012].

It is not uncommon for enterprises to migrate some of their resources to the cloud [Cohen, 2013]. By doing so, the resources become more easily accessible for everybody involved and benefit from a plethora of new capabilities that the web has to offer.

2.1.2. Benefits of migrating to the cloud

Many of today's companies have software resources that need to be shared amongst their employees. These resources could be anything from servers and storage to actual applications, whereas the latter applies to AMA mentioned earlier. Since the cloud is all about connectivity and content distribution, the benefits of migrating desktop applications to the cloud are clear [Group, 2011].

Moving an application to the cloud often means centralizing the resource both physically and administratively, thus lessening the overall IT management burden on the employees. The process is in many ways similar to outsourcing the IT resources of a company to a third party vendor [Chou, 2015].

Another important benefit of cloud computing is scalability. If a company hires new staff that will need access to the software resources, all the new employees will need is a web browser and connectivity to the centralized resource. Compared with the more traditional desktop approach where the new employees would have to make sure they have compatible hardware for the software and then install it locally, the benefits are clear.

Yet another quite descriptive comparison of desktop versus cloud computing that emphasizes the benefits of the latter is the process of upgrading an application. With the desktop approach, every client computer needs to be upgraded locally with the latest software. Thanks to the more distributed nature of cloud applications however, the upgrade only needs to take place once on the server itself.

An environment using desktop applications typically operates as to the left in figure 2.1 below. Every user is operating a desktop application (app.) running on a computer which is communicating in a network of computers.

In the right figure in picture 2.1 below, every user gains access to the application through their computer (denoted as client here). This centralizes the entire workload.

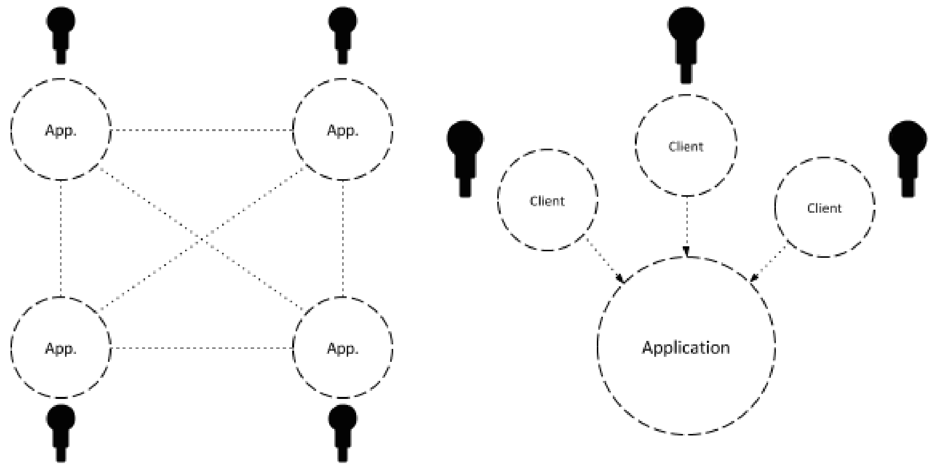


Figure 2.1: A network of desktop computers (left) and an application used as a shared resource (right)

Developing a web application is not very different from developing a website. In fact, a web application is basically a website with the exception that it manages more logic in order to act more like a high functionality application.

2.1.3. Creating dynamic web applications with AngularJS

As mentioned in the background section, originally almost all web pages were static [O'Reilly, 2007]. They were used to simply display information, very much in the same way a newspaper or magazine does.

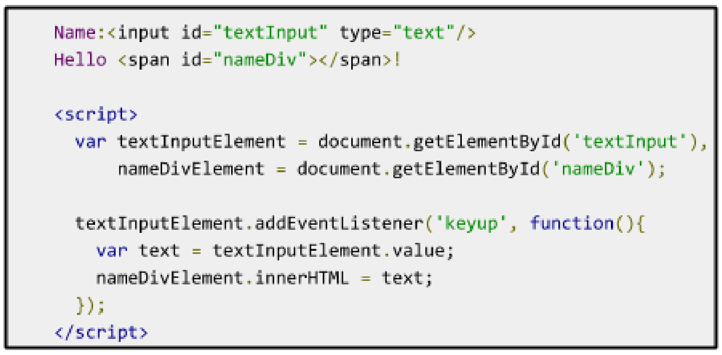
Many desktop applications have to be able to do more than just display information to the user and therefore some problems might arise when trying to migrate a desktop application to the cloud. There is no point in doing such a transformation if important features of the original application are lost in the process.

For a web application to work similarly to a desktop application it needs to be dynamic, meaning that the content displayed to the user can change depending on the input the user gives. A common way to achieve this is to use JavaScript, a scripting language that has become extremely popular amongst all different kinds of web sites. In fact, in 2011, 93% of the world's 500 most popular web sites were using JavaScript [Brown, 2012].

JavaScript is mainly used on the client side of web applications [Brown, 2012]. The code gets executed by the user's web browser and its JavaScript engine. Scripts can be embedded within a normal HTML document and are, amongst other things, used to dynamically manipulate the so called Document Object Model (DOM) that describes the content of the web page. This way it is possible to remove, add and update content that is displayed to the user on the fly.

A popular JavaScript framework is AngularJS, also known as just Angular [Google-Trends, 2015]. Amongst other things, Angular allows for easy two way data binding. If the data changes in one place the change is also reflected in all the elements that references it. Angular and its features are further discussed in later sections. Angular makes use of a JavaScript framework called jQuery that, amongst other things, simplifies the process of programmatically traverse and manipulate the DOM. Consider the following:

Assume that a site should show a message to a user that enters his/her name in the web page. The message should say "Hello" followed by the name that the user inputs. This can be done with or without Angular. Without Angular the code would look something like in figure 2.2.



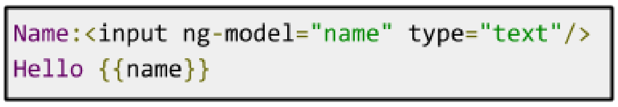
```
Name:<input id="textInput" type="text"/>
Hello <span id="nameDiv"></span>!

<script>
  var textInputElement = document.getElementById('textInput'),
      nameDivElement = document.getElementById('nameDiv');

  textInputElement.addEventListener('keyup', function(){
    var text = textInputElement.value;
    nameDivElement.innerHTML = text;
  });
</script>
```

Figure 2.2: HTML and JavaScript code for displaying "Hello" followed by an entered name

This quickly gets muddled and can be hard to read at first sight. However, with Angular the above example can be solved as seen in figure 2.3 where one states that the model which the input field should update is “name” and then bind the value of “name” by enclosing it within curly braces. This notation is used to tell Angular that whatever is inside the braces should refer to the dynamic value referred to as “name”. It is clear that Angular simplifies a lot of things.



```
Name:<input ng-model="name" type="text"/>
Hello {{name}}
```

Figure 2.3: Displaying "Hello" followed by an entered name using HTML and AngularJS

2.2. UX and HCI

The purpose of this section is to research more thoroughly into the subject of user experience enhancement. In order to improve the workflow of creating questionnaires, the user interface had to be made more intuitive and logical than that of the legacy software.

2.2.1. Background

Human Computer Interaction (HCI) and User Experience (UX) have existed as long as the possibility to interact with computer systems. In 1968, the computer scientist Douglas Engelbart presented his ideas of matching capabilities of humans and computers [Barnes, 1997]. This event played a major part in the history of computer science and is considered to be the beginning of the evolution of Human Computer Interaction [Fischer, 2001].

2.2.2. What is HCI and UX?

HCI is a research field where studies are centered around the usability and design of computers. Research in HCI has primarily been focusing on interface issues in order to create systems that are more easy to interact with [Fischer, 2001]. On the basis of this, the main area of investigations concerning HCI has been in regards to development of rules and criteria for designing intuitive graphical user interfaces (GUI). The reason for this research is to improve the UX and to lower the learning curve for the end user. Since research concerning GUI has evolved, interface problems have been better understood.

The term UX has previously been tightly connected with HCI but UX has grown in importance and can be seen as a positive side of HCI [Hazzenzahl, 2006]. Where HCI primary focuses on the creation of systems that are easy to interact with, UX aims to give the end user a pleasant experience. This experience would allude to the user emotions, and try to prevent negative feelings.

To offer a pleasant experience for the user, the design of the GUI is highly valued. For example, it takes no longer than 50 ms for a user to appraise a web page [Lindgaard, 2006] and therefore the visual experience has to be aesthetically appealing. Aesthetics plays a major role in how users obtain information [David, 2010].

To sum up this section, a computer system needs to be easy to use, allude to the end users emotions and be aesthetically appealing.

2.2.3. HCI and UX importance

Ever since Engelbart and his team presented their vision in 1968 [Barnes, 1997] HCI and UX have grown in importance for software companies all over the world. The basis for HCI at the very core of the subject has not changed. The graphical user interface sets expectations on functionality and provides a way for the user to access said functionality.

To quantify the importance of HCI in software projects, a real and relevant method of measurement must be used. One such measurement method that arguably works despite the abstract and subjective nature of UX is the return of investment, ROI, method [Beattie, 2014]. The basis of ROI is to measure how many dollars worth every dollar invested returns. To achieve realistic ROI all relevant aspects need to be taken into account. Some of these aspects are costs in development, end user response and and the amount of post-deployment service needed.

At the start of a new project, focusing on the user experience leads to clearly defined requirements. This makes it easier to both implement the front end and to provide the back end functionality. This aspect can be so advantageous that the development takes up to half as much time. Other important tools are personas and wireframing. Studies have shown that they result in more realistic time estimates, reduced amount of requests for clarification and that they also lower the need for rework and bug fixing after deployment [Spillers, 2014].

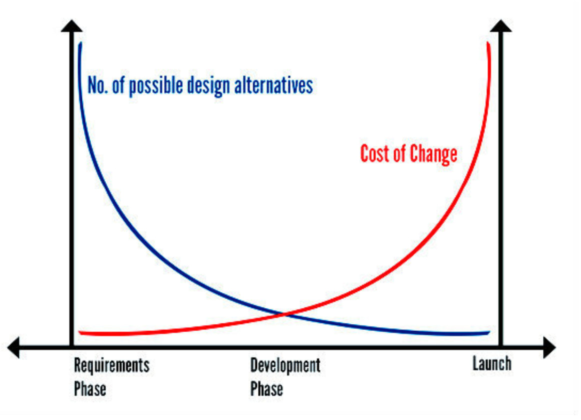


Figure 2.4: The importance of proper UX development can be represented with the 1:10:100 rule. Spend $1 on research, $10 to change design or $100 to change something already in development. Picture from [Spillers, 2014]

newpage

Examples where UX investments paid off are when McAfee experienced a 90% reduction in support costs after increasing their usability testing [Spillers, 2014] and that Airbnb went from failure to a billion dollar company after listening more closely to customer feedback on their service [Gebbia, 2013]. Data shows that on average 10% of the development budget is spent on this area and that the ROI can range between \$10 and \$100 for every dollar spent [Lai, 2014]. The average return of investment after a redesign is 83% [Nielsen, 2008].

2.2.4. Problems with High-functionality applications

In the corporate world, the kind of software used is often advanced and complex. Usually these systems are directed to professional users and, most commonly, have a high learning curve. These complex system are called High functionality applications (HFA)[Fischer, 2001]. The ambition with these applications is to model the existing world as accurately as possible. These ambitions often lead to applications that are unnecessarily advanced and unintuitive. Companies developing this kind of software more often mention that the professional users using these applications will learn to use them after spending some time with them. How complex a HFA becomes depends on the amount of features that needs to be implemented. Fischer mentions a hypothesis in his article that reads as follows:

“If you asked 100 different people what features they would like to have in a particular application, you would end up with a very large number of features.”

The essence of this quote is that it is hard to create software that offers a pleasant experience for the user that also satisfies a large group of users. To create a HFA that satisfies the majority of the users but still takes design into account, the application has to get rid of unused functionality. This means that they cannot get in the way for features that are being used more frequently. Still, the features that are hidden have to be easily accessible when they have an actual value for the user. The last design issue that needs to be taken into account is that the core features need to be easy to use and learn.

Corporations working in the financial sector, such as AMA, have more recently taken user experience into account regarding their business systems[Kumar, 2014]. The article suggests that this change in attitude depends on three factors. The first factor is the changing conditions of technology. In the 21st century the new and modern technology first reached private individuals. New phones, computers and applications are first acquired by individuals, rather than corporations. This is different from the attitude just a decade ago. Back then, new technology reached corporations before consumers. This change tells us that since consumers adapt new technology more rapidly, they also get the first sight of new user experiences. Suddenly, the software used at work feels outdated and not as aesthetically appealing.

Another factor that signals a major change in attitude regarding business software is that the barriers between work and private life are fading away. The demographic change of workers in the concerned industries has to be considered [Kumar, 2014].

Many of the employees of today's corporations have grown up in a society where computers have been a natural element of their everyday life.

2.3. GUI Evaluation methods

UX design in software is still a very young field. There are many different methods to evaluate a particular application and all of them have different advantages and disadvantages.

2.3.1. Heuristic Evaluation

Heuristic Evaluation is a well recognized and adopted method of evaluating software in regards to how well its Graphical User Interface(GUI) manages to empower the user [Nielsen, 1994]. The method involves a set of evaluators that evaluate the software by discussing how well it conforms to a set of predefined principles. Even though these predefined "heuristics" can be created specifically for a certain application, it is much more common to go by some of the already established ones [Nielsen, 1995a]. One of these are the Nielsen's heuristics [Nielsen, 1995b]. They were developed by Jakob Nielsen in 1995 and is still the most used Heuristic evaluation method. This method covers a large set of issues that has to be addressed in creating a great user experience.

The ten heuristics are as follows:

"Visibility of system status" [Nielsen, 1995b]
The user should be fully aware of what task the system is performing.

"Match between system and the real world" [Nielsen, 1995b]
To fully cater to a wide user base, the application should communicate by means that are easily recognized by the user rather than using language or performing actions that are too technical.

"User control and freedom" [Nielsen, 1995b]
The user should never feel as if a choice can't be reversed. The user has to be able to perform erroneous actions and not be punished by them, but rather be offered to undo.

"Consistency and standards" [Nielsen, 1995b]
The application should follow a uniform standard when it comes to all forms of user interaction. This makes navigating and interaction less cumbersome for the user.

"Error prevention" [Nielsen, 1995b]
Prevent the user from performing unintentional errors.

"Recognition rather than recall" [Nielsen, 1995b]
The user should be able to recognize events within the software to easily act upon them, this is far superior to requiring the user to memorize them.

"Flexibility and efficiency of use" [Nielsen, 1995b]
Allow an expert user to engage in a faster and more efficient workflow.

"Aesthetic and minimalist design" [Nielsen, 1995b]
By getting to the point and eliminating unuseful information the user can engage more thoroughly in the message that you, as a developer, are trying to convey.

"Help users recognize, diagnose, and recover from errors" [Nielsen, 1995b]

When errors occur, try to enable the user to handle these himself by providing easily understood instructions.

"Help and documentation" [Nielsen, 1995b]
Help and documentation should only be presented for the sole purpose of enabling the user to fully understand and operate the application. Therefore it is important that the documentation is easily understood and absorbed.

2.3.2. Cognitive Walkthrough

Cognitive Walkthrough is, just as the heuristic evaluation, a method of analysing the user interface of an application or product[Spencer, 2000]. The walkthrough aims at identifying actual flaws of the user interface down to the exact graphical component rather than providing a general understanding of the weaknesses of the application, which is what the heuristic evaluation provides.

The method involves evaluators that will evaluate the software and moderators that lead the proceedings (moderators can also be evaluators themselves). Performing a cognitive walkthrough is based around a set of questions and a certain task that are predetermined by the evaluators [Spencer, 2000]. The questions can be of any nature and of any quantity, although the goal is to properly identify the flaws of the interface, and the task should put the evaluators through a part of the application. An example of a task would be to send a text message on a phone, if the evaluators were to evaluate a text messaging app on a phone.

Even though one can construct questions freely, there are already many sets of questions available from researchers within the field. They range from fast and streamlined approaches such as Spencer's “Streamlined cognitive walkthrough” [Spencer, 2000], that aims to simplify the process and make it more accessible, to more broadly adopted versions such as the one described in the third edition of “Interaction Design - Beyond Human Computer Interaction” [Rogers, 2011].

The sets of questions described there are as follows:

1. “Will the correct action be sufficiently evident to the user?" [Spencer, 2000] (Will the user know what to do to achieve the task?)
2. “Will the user notice that the correct action is available?" [Spencer, 2000] (Can users see the button or menu item that they should use for the next action? is it apparent when it is needed?)”

3. “Will the user associate and interpret the response from the action correctly?" [Spencer, 2000] (Will the users know from the feedback that they have made a correct or incorrect choice of action?)”

3. The legacy software and the previous workflow

In this section, the shortcomings of the legacy software and the workflow that comes with it are discussed. Also, methods for evaluating both the legacy software and the new web application are presented along with evaluation results.

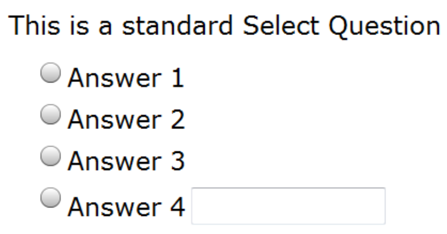
3.1. The legacy software

The key feature of the legacy software is the ability to create questionnaires. These questionnaires are used in research studies, and are uploaded to the Internet where research subjects can be interviewed about their opinions concerning a specific issue. In the legacy application it is possible to use six different components for interviewing the research subjects. Each component can be seen as a set of different question types that can be used to customize the questionnaire. The components that can be accessed are the following:

- General Computer Interviewing (CiW)
- Choice-Based Conjoint (CBC)
- Adaptive Choice-Based Conjoint (ACBC)
- Adaptive Conjoint Analysis (ACA)
- Conjoint Value Analysis (CVA)
- Best/Worst Scaling (MaxDiff)

In this project, the primary focus has been on CiW. CiW consists of the main elements that are used to build questionnaires. They provide the possibility of creating questions, and there are 11 different question types to choose from. All of them are not relevant for this project, but the essential ones are the following:

- Select Question. Used to ask the research subject a question and give him or her a set of answers to choose from.


This is a standard Select Question

Answer 1

Answer 2

Answer 3

Answer 4

Figure 3.1: A Select Question

- Numeric Question. Used to ask the research subject to type a number as a response to a question.

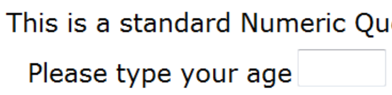


Figure 3.2: A Numeric Question

- Open-End Question. Used to give the research subject the possibility to answer a question in a text area.

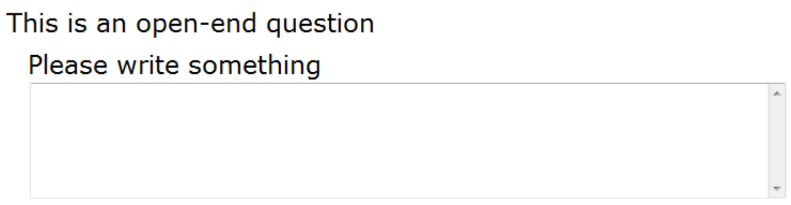


Figure 3.3: An Open-End Question

- Grid Question. Used to ask a series of questions where every question has the same set of answers.

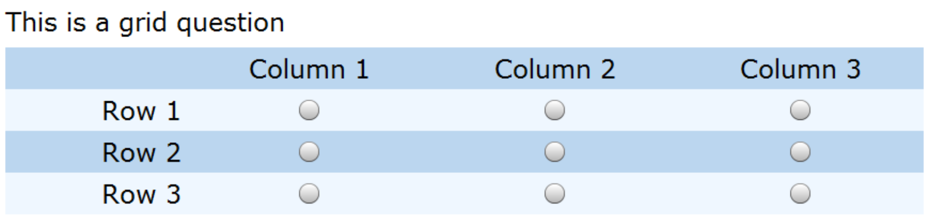


Figure 3.4: A Grid Question

Apart from the different question types, the legacy software also contains features such as skip logic. This feature makes it possible to give questionnaires a dynamic behaviour. With skip logic, the contents of the questionnaire can be seen as a tree structure, where different paths show different questions. As seen in the picture below, skip logic can be used in different ways referred to as "skip logic patterns". For example, it is possible to specify that some questionnaire items should be skipped all together, if the subject gives a certain answer to some previous question. Refer to figure 3.5 below for clarification.

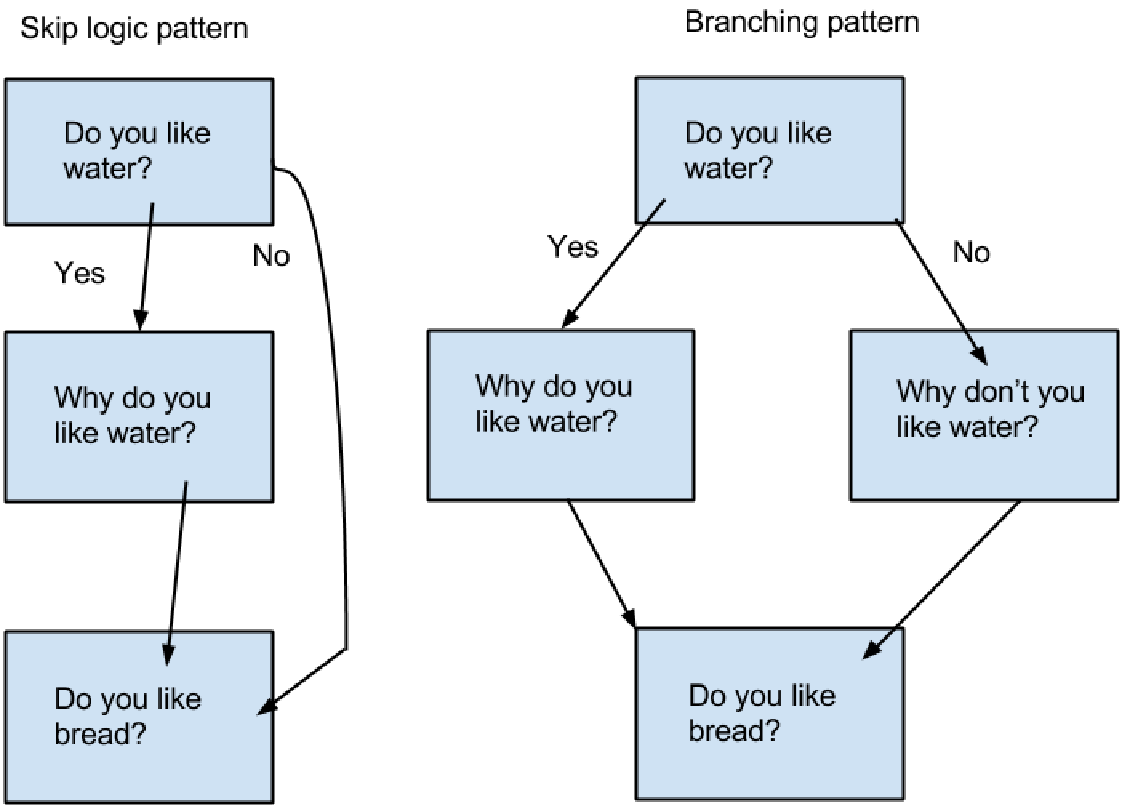
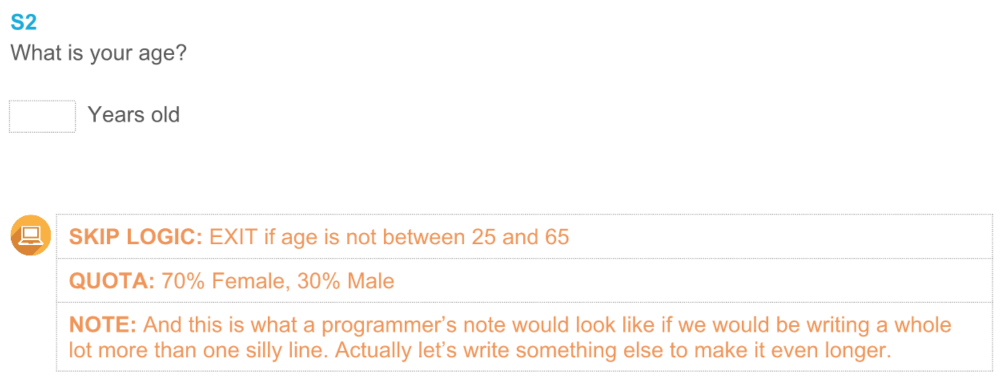


Figure 3.5: Different ways of using Skip Logic

3.2. Workflow at AMA

AMA is a multinational company working with quantitative and qualitative research studies, providing the study results to research agencies and consultancy firms. In order to do research and analysis for their customers, a software that supplies the relevant services is essential for the company. The purpose of the legacy application is for AMA to create studies about specific topics. These studies will later be sent out to the chosen research subjects. The creation of questionnaires is one part of the services that AMA provides their customers, but in this project the focus will be delimited on this.

The delimited process regarding AMA's services is centered around a repetitive pattern of how to create and visualize questionnaires in a comprehensible way for customers. The workflow at AMA consists of two major parts. AMA begins by producing a text document containing all relevant information about the questionnaire, including the structure of the skip logics and all the questions. When the text document is considered to be complete it will be sent to the client for feedback.



S2
What is your age?

Years old

SKIP LOGIC: EXIT if age is not between 25 and 65

QUOTA: 70% Female, 30% Male

NOTE: And this is what a programmer's note would look like if we would be writing a whole lot more than one silly line. Actually let's write something else to make it even longer.

Figure 3.6: Example on what a Numeric Question looks like in the text document.

The second step in the workflow at AMA, is the creation of the questionnaire in the legacy application, using the text document mentioned above as a template. This questionnaire will also receive feedback from clients when needed, and according to this, changes would be made until an approval from the client. At this point the process would be considered done, and the survey produced in the current software will be sent to its research subjects. Below is a chart describing the entire workflow.

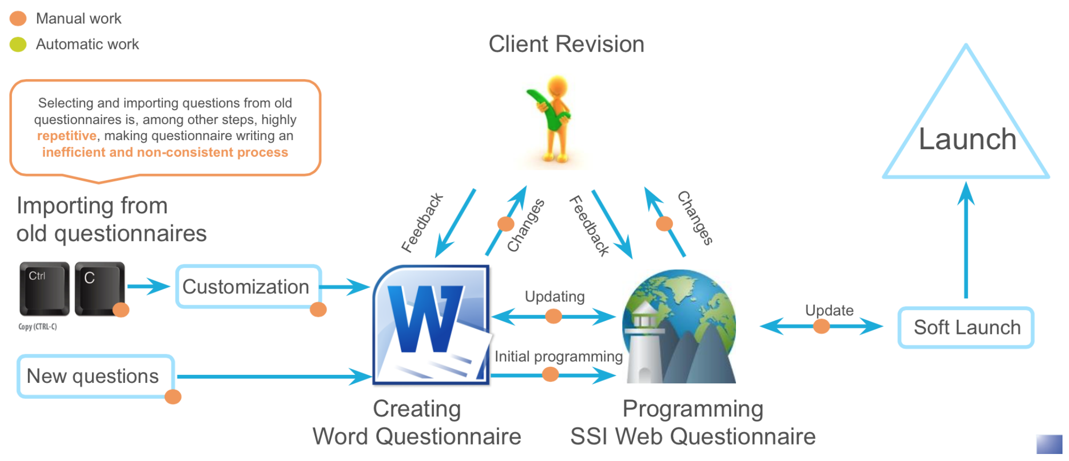

Manual work
Automatic work
Client Revision
Launch
Selecting and importing questions from old questionnaires is, among other steps, highly repetitive, making questionnaire writing an inefficient and non-consistent process
Importing from old questionnaires
Ctrl
C
Copy (CTRL-C)
Customization
New questions
Feedback
Changes
Feedback
Changes
Updating
Initial programming
Update
Soft Launch
Creating Word Questionnaire
Programming SSI Web Questionnaire

Figure 3.7: The workflow at AMA

3.2.1. AMA's opinions on previous workflow and software

According to AMA, this workflow consists of highly time consuming work. The construction of the questionnaire itself is particularly tedious due to the iterative process towards the clients that created prolix cycles of feedback. AMA also mentions that the creation of questionnaires is far from ideal. There are no possibility to reuse old standard questions, which implies a great deal of repetitive work. This results in a time-consuming process without value for the company.

According to the interview with representatives from AMA, found in appendix A.2, the overall impression of the legacy application is that it feels old and does not give the user a pleasant experience. It contains many features but few of which are actually used on a daily basis. The relevant features can be hard to find and use and the software does not offer a pleasurable overview of the survey.

3.3. AMA's visions for the new workflow

AMA mentions in an interview, found in appendix A.2, that they have great visions of how their workflow should be. This vision include a new software that addresses the shortcomings of the previous workflow. AMA mentions in the interview:

"The overall objective is to provide a software that makes the creating process of clients' questionnaire and surveys more efficient, standardized and of higher quality."

To do this, the two main parts of the questionnaire creation process have to be merged into one. The new software would be able to automatically generate the text documents from a questionnaire. This means ultimately that when the client has received the text document and agreed on the content, the survey would also be approved. Figure 3.8 shows what the new workflow would look like.

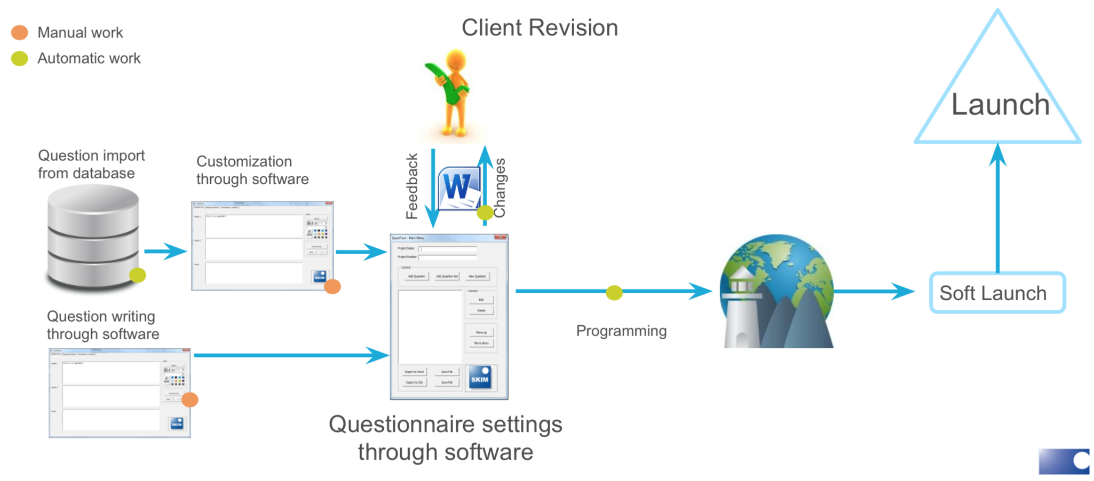

Manual work
Automatic work
Client Revision
Launch
Question import from database
Customization through software
Feedback
Changes
Question writing through software
Programming
Soft Launch
Questionnaire settings through software

Figure 3.8: The desired workflow at AMA

AMA also mentions visions and ambitions about the appearances of the new software. First of all, it should offer a clear overview of the questionnaire, and every survey item within the questionnaire should be easy to find. The questionnaire overview should look like the text document.

Another issue that AMA wants to eliminate, is the amount of unused features that the current software have. The most used ones should be easy to access and use.

The more seldomly used features should remain easily accessible without being in the way for the more frequently used ones.

The last vision that AMA has for the new software is the ability to reuse questions. A vast amount of time is spent on recreating standard questions for every new questionnaire. The objective is to have a central database where sets of questions can be saved. Since the database is shared, a large volume of questions would be easily accessed by all employees.

AMA's aim is to replace the usage of word processors and the current software completely. However, the first step is to create a software that fulfills the most basic visions, such as text document generation and database support mentioned earlier. Additional settings that the new software does not cover will need to be done in the legacy software.

The ambition with these changes is to encourage a workflow that meets the client's demand on faster delivery. The new workflow will make the process less time consuming and therefore more cost effective.

3.4. UX Evaluation process

To fully understand the negative and positive aspects of the legacy application, user experience evaluations were performed.

3.4.1. Cognitive Walkthrough of current software

The cognitive walkthrough concluded that the legacy software is quite divided and non-uniform. There are sometimes plenty of ways to perform a task while there might just be one at other times. Icons and buttons are seldomly uniform and this causes quite some confusion. Even though this is the case, the legacy software generally enables the user to perform the wished task.

The main problem with the legacy software is the constant need for opening new windows whenever the user wishes to perform a new action. This makes it very cumbersome for the user to navigate between different features.

"Will the correct action be sufficiently evident to the user?" [Spencer, 2000]

> Often, the next step is not intuitive. The user has to perform a set of actions to achieve the desired goal rather than cutting straight to the point. An example of this would be that a user wants to perform a change to a question. Rather than enabling the user to easily correct something within a question from the overview window, the user has to navigate a set of ,at times confusing, menus to achieve the initial goal.

"Will the user notice that the correct action is available?" [Spencer, 2000]

> Most of the time, the option is quite evident. Even though icons and buttons does not always conform to the standard, they are often clearly visible.

This is mainly due to the fact that the legacy software is trying to display all the available features rather than highlighting the most important ones. The result of this is a cluttered interface that might be hard to navigate but nevertheless allows the user to find the correct button eventually.

"Will the user associate and interpret the response from the action correctly?" [Spencer, 2000]
This turned out to be the field in which the legacy software performed the best. Since it often opens a new window for each feature, the user instantly understands that a new task has been performed by the application. The only problem can be that, as previously mentioned, the application is not uniform in the way it conveys information. Therefore it can sometimes be hard for the user to tell if the new window that appears is the correct one for what the user might wish to accomplish.

3.4.2. Heuristic Evaluation of the legacy software

The conducted heuristic evaluation concluded that the legacy software contains a vast amount of graphical interface and user experience related shortcomings.

Even though the application showed promise in some isolated areas, the majority of the opinions of the evaluators were of the criticizing nature rather than of the praising kind.

Positive aspects

One major positive feature that permeates the whole application is also, in some sense, negative. The legacy software displays a pop up when a new feature is initiated (Example: Editing a question, Entering skip logic etc. etc.). This helps the user to understand what the application is doing and goes in accordance with the first out of the ten Nielsen heuristics.

The application is uniformly designed in accordance with the third Nielsen heuristic which takes into account the freedom of the user. The legacy software allows for the user to fully revert the actions that is taken. This is done using regular "Cancel" and "Close" buttons on many views.

The third positive aspect of the legacy software is the error messages. These messages are clear and straight to the point, describing in detail what went wrong in a way the user will understand. This falls under the ninth Nielsen heuristic which describes the importance of having easily understandable error messages.

Negative aspects

The application boasts a long list of wrongfully designed components in regards to the user experience.

As stated earlier, the way that the legacy software displays pop ups for most new features makes the user aware of what the application is doing. But on the other hand, the workspace gets very cluttered and there is no sense of overview since the user is constantly navigating back and forth between different levels of the interface.

Something that is a major issue for the original application in general is the lack of a uniform design when it comes to buttons and icons. These interactive components are often combined and put in a suboptimal order and position in regards to each other. Also they do often times not match the standard usages within other applications (an extreme example of this would be that the magnifying glass icon does not enable a search function, something that is considered a universal standard[Nielsen, 2014]).

The positioning and usages of these buttons and icons contribute to the cluttered and cumbersome interface of the application. Often the application tries to fit in every aspect of a feature within the same view, something that results in the view being filled to the brim with graphical components. This also results in important features not being given enough space to stand out and therefore blending in with less important ones.

Another issue with the application is the technical nature of the documentation. A large quantity of the text components of the user interface includes technical language that a novice user would not be able to fully understand.

There are also two major parts of the application that causes great confusion to a novice user.

The first one is the Skip Logic view. Creating skip logics is far from easy. Apart from the somewhat minor defects of the graphical interface within the skip logic view, the actual skip logics themselves are very cryptic and very hard to fully understand without the guidance of an expert user. The skip logics are composed by using an odd and unintuitive programming language that takes quite some time to get used to.

The issues regarding the creation of skip logics is very similar to that of the constructed list creation process. The list creation view is both cluttered, unintuitive and even features some bugs that crawled their way through the development process. But just like the skip logic feature, the creation of constructed list also requires programming skills. The constructed list is created by constructing a small PERL script that generates the list. This goes far and beyond what is supposed to be needed by the user and hence incapacitates the novice user.

3.4.3. Evaluation Summary

The evaluation of the legacy software concluded that the program contains a broad range of critical errors in regards to the user experience. It is cumbersome to operate even for the experienced user, something that flattens the learning curve for a new user.

The legacy software provides a non-uniform approach to graphical user interface design and goes against many rules of user experience design. It shows great promise in what features it provides but the graphical user interface burrows these features in a layer of unnecessarily complex and confusing maze of buttons, windows, icons and more.

The main things that have to be done are the following:

1. Provide a better overview of the Survey and the questions it contains
2. Prioritize the functionalities of the application and make these easier to access
3. Eliminate the use for programming knowledge
4. Prevent errors, this could be done by checking text fields so they aren't empty for example
5. Provide a more uniform approach to button positioning, phrasing and the use of icons
6. Simply work towards making the application less cumbersome and thereby both increase the efficiency of the workflow and also make it easier for novice users to adopt it.

3.5. The prototype

According to AMAs' opinions and the results received from both the cognitive walkthrough and the heuristic evaluation, GUI drafts were produced to give an outline on how the new software should look like. The ambition with the prototype, called "QuestBuilder", is to create an application that produces a text document and a survey file. The sole purpose of this project is to aid the structuring of a new and more efficient workflow for AMA consultants.

The features that were ultimately implemented in the final software are:

- Select Question
- Skip Logic
- Constructed Lists
- Database Export
- Text document generation
- Survey file Export
- Survey file Import
- Pagebreaks
- Skip logic notes
- Quota notes
- General notes
- Request to client notes

The back end supports all CiW question types.

The major goal is to create a software that gives a good overview of what is actually being created. When doing this, the principles mentioned in 2.2.4 will be taken into account [Fischer, 2001]. The principles says that features that are commonly used need to be clearly visible and easy to access. The ambition with QuestBuilder is to make the question creating process more intuitive, since that feature plays a key role in the process of constructing questionnaires. When dragging a question type into the center stage, the question will be created and visualized for the user. The user edits the question in real time, and the result is displayed in the same way as in the text documents. It should also be possible to arrange the questions, giving the user the possibility to place the questions in the preferable order. Figure 3.9 shows an early draft of what the center stage could look like.

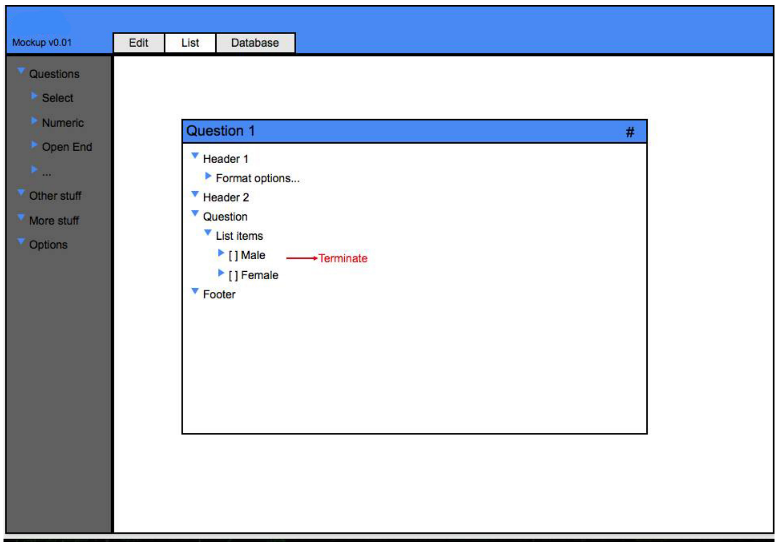


Figure 3.9: Early draft of the center stage with one question added to the survey

As mentioned in section 2.2.4, HFA of good quality needs to have seldom used features easy to access when they have an actual value [Fischer, 2001]. The goal of QuestBuilder is to collect every infrequently used feature, which in this case usually are advanced settings, into a unique separate view for every question type. This advanced view should be easy accessible for the user with a single mouseclick from the center stage. The advanced settings will be in the shape of a popup that covers almost the whole screen. By doing this, the ambition is to offer a way to reach the advanced settings, but yet to remain in control of the questionnaire in the making. In Figure 3.10 the ambitions with the popup window will be shown.

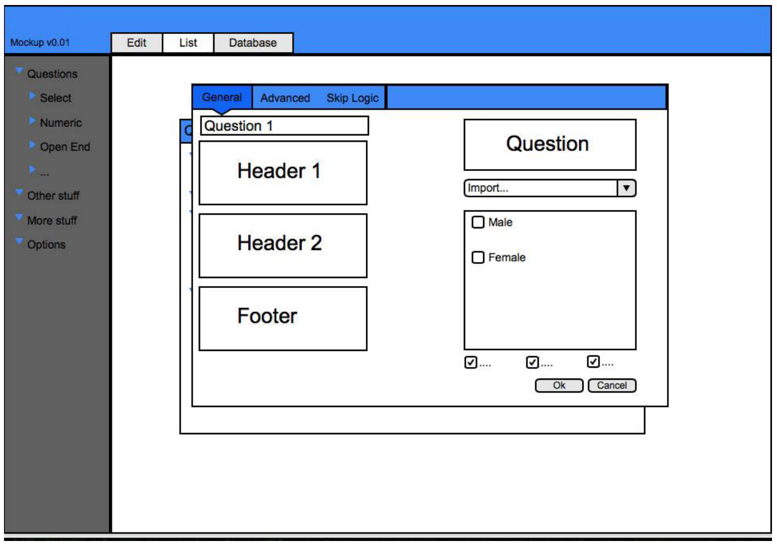


Figure 3.10: Early draft of advanced settings

Previous evaluations showed that the method of creating skip logics is far from optimal. A major issue is that it is hard to understand the logics created, and how it will evaluate in the end. Since everything is based upon text and not actual visuals, the user needs to have a deep understanding of how skip logics work. The goal of skip logics in QuestBuilder is to eliminate these obstacles of creating skip logics, but yet to keep the complexity that the legacy software offers. By doing this, the thoroughgoing topic of always having a general overview of the questionnaire will impregnate skip logic as well. The solution is to give the user control over how the skip logic looks like, and to always show the conditions that will activate the skip.

To create constructed lists is far from ideal, according to the evaluation in section 3.4. The ambition with the new application is to make it more intuitive to create lists, even if the user has no experience of PERL scripting. The ambitions are similar to ambitions of skip logics; to eliminate the obstacles. However, constructed lists have several additional features that are hard to simplify. A combination between accessible features and PERL scripting is therefore considerable.

By developing the software according to these ideas, the belief is to offer a HFA designed according to the ideas mentioned in section 2.2.4. By emphasize the core features and have them easily accessible, the problems of having a large quantity of functionality visible will be eliminated. The advanced features will be reachable by a single button press. This way, they will not take up space from the fundamental features. Lastly, the basic features have to be intuitive to use and learn. In QuestBuilder, the ambition is to provide a logical pattern throughout the whole application and to give the user total control over the questionnaire creating process.

.

At the time of the creation of this publication and evaluation, the prototype is in the final stages of development. The shortcomings and issues addressed here will be taken into account before the final release of the prototype.

4.1. Presentation of QuestBuilder

What follows in this section is a presentation of how the GUI of the new application looks like. All found shortcomings regarding the desktop application, both from AMA and the evaluations made in section 3.4, have been taken into account during the development. The design has continuously been reviewed by AMA in order to tailor it to their needs.

4.1.1. Main View

The main view follows a design pattern called Center Stage [Tidwell, 2015], a pattern used to put focus on the most relevant parts of the software but still offer access to other features. In QuestBuilder, the user can access every type of question in a side panel. The question can subsequently be dragged into the center stage in order to add that element to the survey. Every survey item, including page breaks, operates according to this standard.

Navigation to different parts of the web application, is located in a menu bar at the top. When browsing the main view, a second bar is positioned at the top of the center stage. This is where file options such as importing and exporting survey files as well as exporting text documents is located. This is also where the toolbar with text editing options, such as bold formatting, will appear when it is needed.

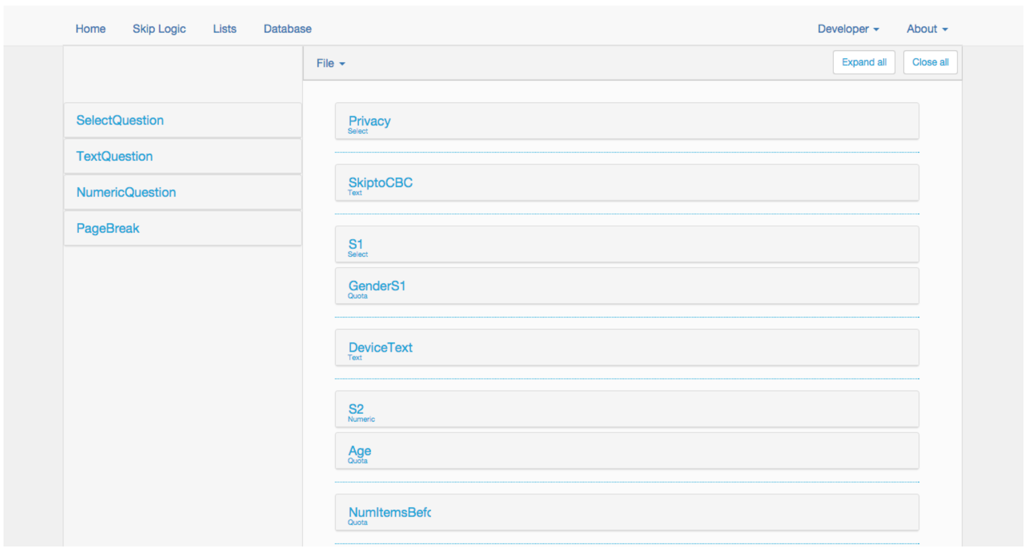


Figure 4.1: Mainview of the QuestBuilder application

4.1.2. Select Question

Questionnaire items such as "Select Question" are added to the questionnaire by dragging the desired item from the sidebar to the center stage.

It is possible to edit a question item in the center stage. When a Select Question is created, the user will be forced to specify the name of the question. The question will then be visualized according to how it will look in the text document generated from the application, with the same styling and word wrapping.

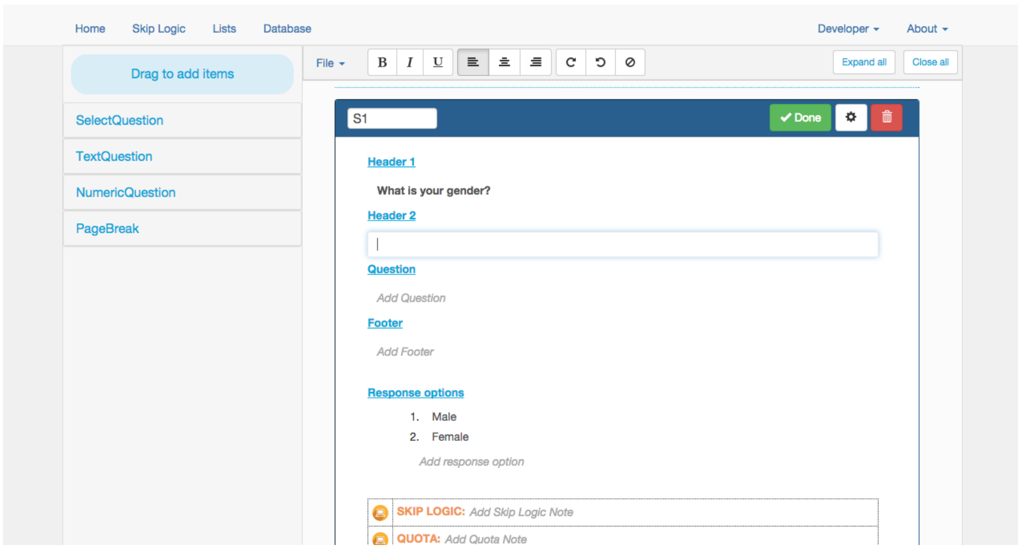
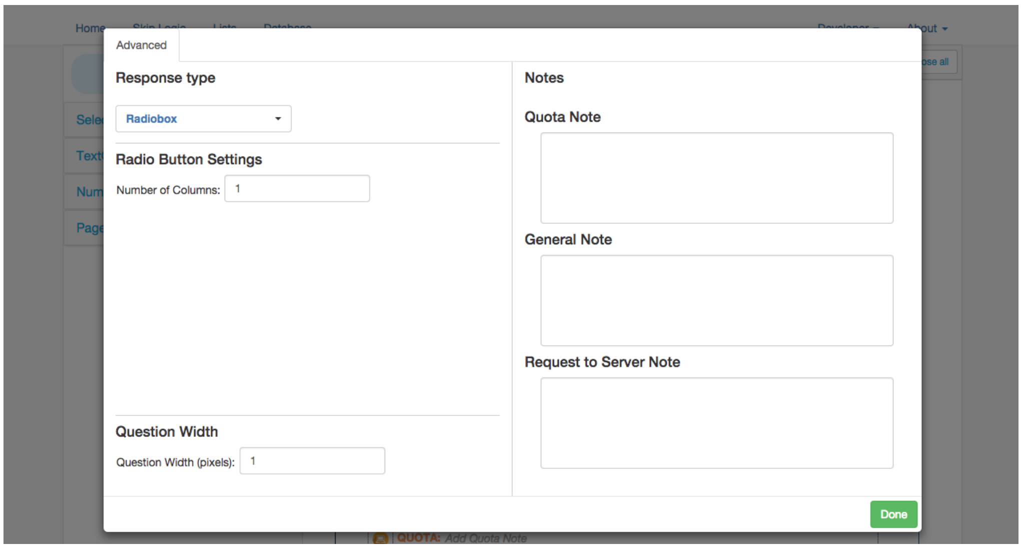


Figure 4.2: Basic Select Question editing

A select question has an advanced view. It is accessed by pressing the settings button at the top corner of every question item in the center stage. This view is a pop up window, which lays on top of the center stage.

The advanced view includes settings that are rarely used. The user can change the response type, and based on which response type that has been chosen, unique settings for that response type will be visualized.



Response type: Radiobox

Number of Columns: 1

Question Width (pixels): 1

Quota Note

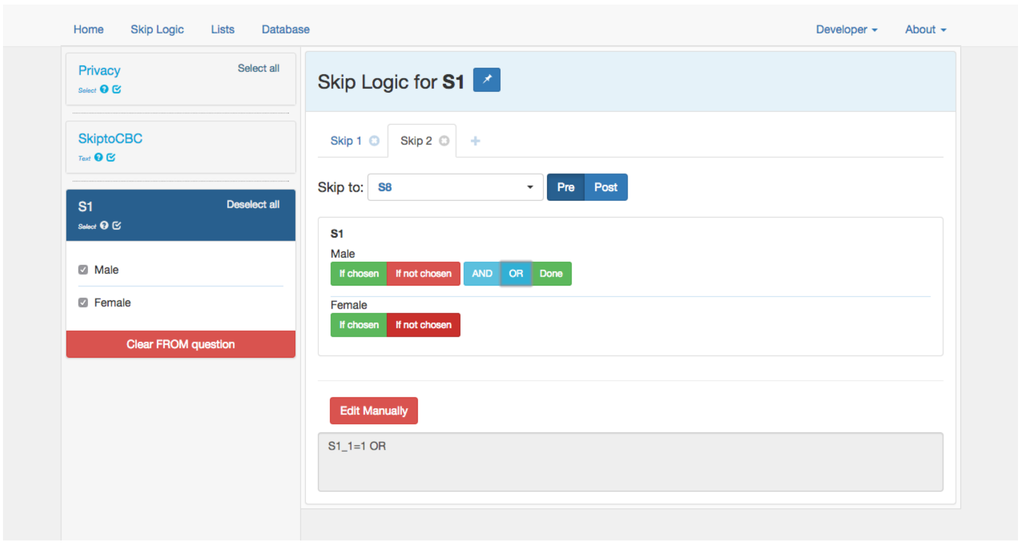
General Note

Request to Server Note

Figure 4.3: Advanced Select Question editing

4.1.3. Skip Logic

The Skip Logic view uses the same center stage pattern as the main view. In the left side panel, the different question types have been replaced on behalf of the contents of the questionnaire. To create a skip logic the user chooses a question from the list from which it should skip from. The skip logic is visible in plain text at the bottom of the center stage, and the user can choose to manually construct a skip logic without pressing the buttons. Since a skip logic can be based on several questions, all questions that precede the question chosen to skip from can be accessed as well.



Skip to: S8

```
S1_1=1 OR
```

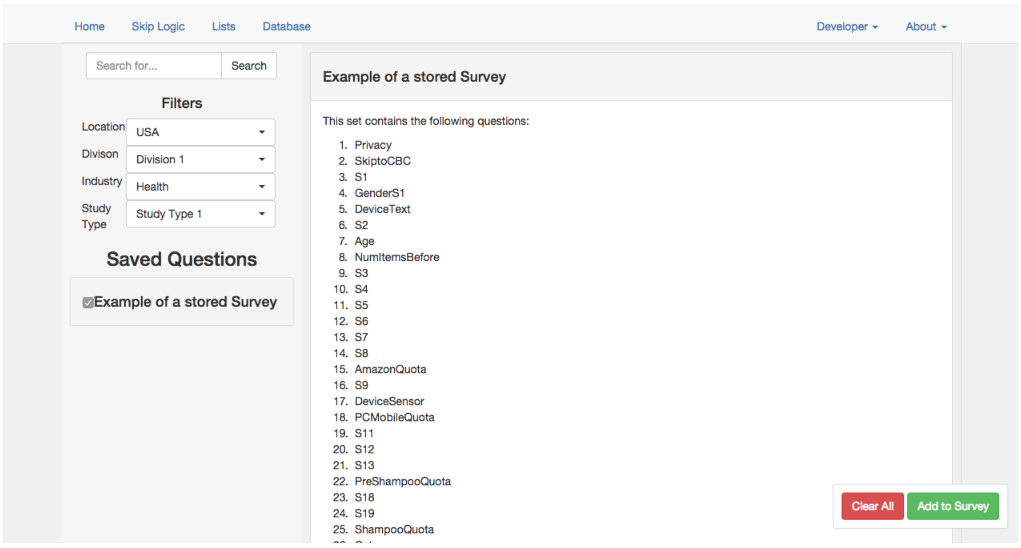
Figure 4.4: Adding skip logic to a question

4.1.4. Database

The possibility of saving questions to a database, is an additional feature that helps improving the workflow at AMA. The database makes it possible to save a set of questions to a shared database accessible by all employees.

From the main view of QuestBuilder the user can unlock the possibility of selecting which questions that should be saved to the database. After the questions has been selected a pop up similar to the settings view will be displayed. In this pop up it is possible to specify the name of the set but also to pick four different tags that AMA has proposed.

When a set of questions have been stored in the database, the users can search for that set and import it to their own surveys. This is done in the Database view. This view is also based upon the Center Stage pattern. Located in the top half of the side panel is a search field, used to search for stored sets of questions. The results are immediately shown in the bottom half of the side panel. The user can sequentially choose the sets to import in the survey. The chosen sets, including their questions, will be visible in the center stage. Ultimately, when all the preferable sets have been selected, the import button can be pressed in order to import them to the ongoing survey.



Location: USA
Divison: Division 1
Industry: Health
Study Type: Study Type 1

Saved Questions

☑ Example of a stored Survey

Example of a stored Survey

This set contains the following questions:

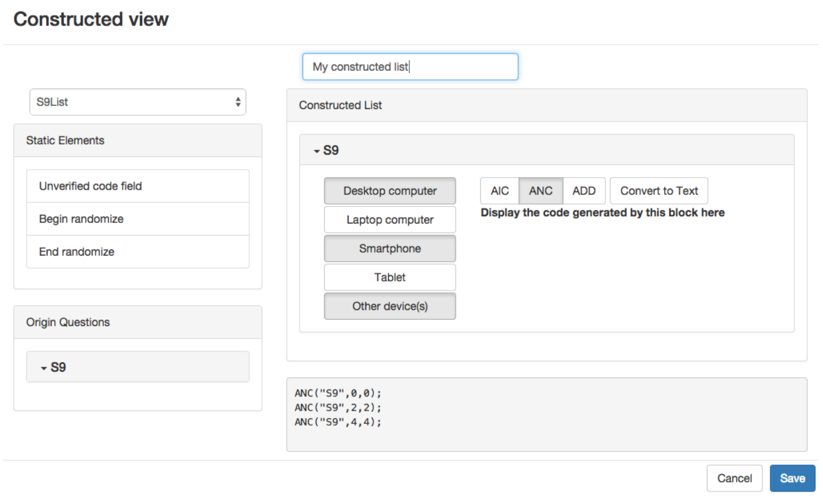
1. Privacy
2. SkiptoCBC
3. S1
4. GenderS1
5. DeviceText
6. S2
7. Age
8. NumItemsBefore
9. S3
10. S4
11. S5
12. S6
13. S7
14. S8
15. AmazonQuota
16. S9
17. DeviceSensor
18. PCMobileQuota
19. S11
20. S12
21. S13
22. PreShampooQuota
23. S18
24. S19
25. ShampooQuota

Figure 4.5: The Database View

4.1.5. Constructed List

The view where the user can create Constructed Lists, is available by choosing “Lists” in the navigation bar at the top. The intention with this view is to make the creation of Constructed Lists more intuitive. Constructed Lists are built by first choosing a parent list, the list that the new constructed one should depend on. The user then drags elements from the left half into the right half. The left half elements are Origin Questions, which are questions created with the chosen parent list. There are also static elements, which at this stage of development can be used to randomize options or to write code in pure text.

By choosing elements from the left pane, and drag them to the right one, the user can build the constructed list. If an origin question has been chosen, the user has the possibility of adding logical operations to the response options. These options are “Add if Chosen” (AIC), “Add if not Chosen" (ANC) and “Always Add” (ADD). These options will generate PERL code that is displayed in the bottom right half.

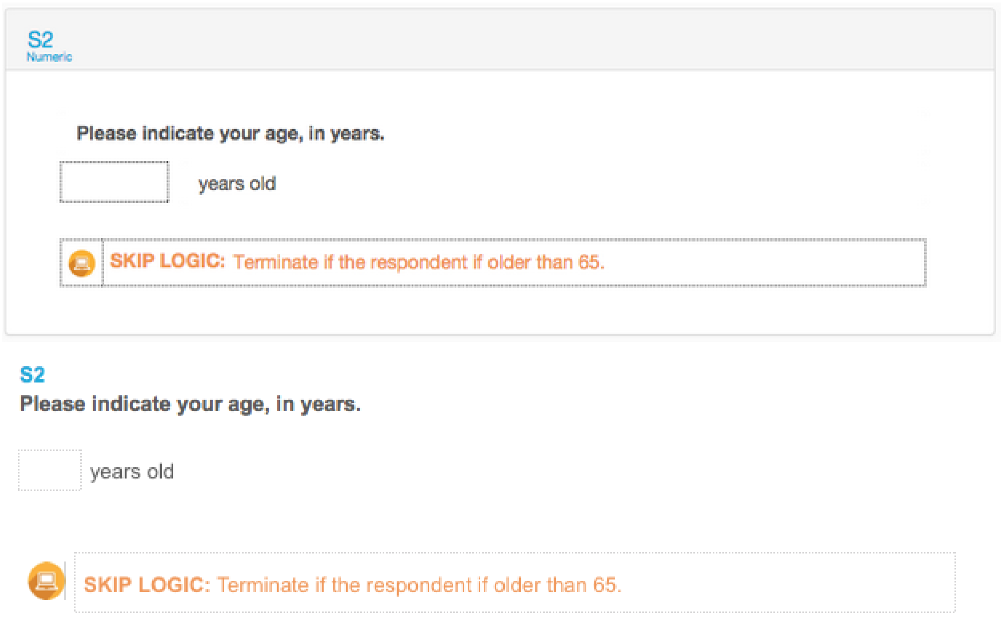


```
ANC("S9",0,0);
ANC("S9",2,2);
ANC("S9",4,4);
```

Figure 4.6: Creating a constructed list

4.1.6. Import and Export

QuestBuilder is capable of importing existing survey files, as well as exporting questionnaires to this file format. It is also able to generate Microsoft® Word compatible documents from the prototype. Questions will have the exact same appearance in the document as it will in the application. This can be seen in figure 4.7 below. This way, the employees at AMA does not need to create a text document and a survey file in two different processes. Instead, both the document and the questionnaire itself can be produced in QuestBuilder simultaneously.



S2
Numeric

Please indicate your age, in years.

years old

SKIP LOGIC: Terminate if the respondent if older than 65.

S2
Please indicate your age, in years.

years old

SKIP LOGIC: Terminate if the respondent if older than 65.

Figure 4.7: A Numeric Question in the main view of QuestBuilder (top) and a Numeric Question in a text document (bottom)

4.2. Cognitive Walkthrough

The cognitive walkthrough of the Questbuilder application uncovered a few instances where the application obfuscated the features, making the user experience confusing and inhibiting an efficient workflow.

Even though there were some flaws, the evaluators general opinion where that the application had improved upon the concepts of the legacy application. This is the result of a number of adopted design principles.

Firstly, the new web application provides a more overview centered approach. The application focuses on putting the content first. This is done by providing all the produced content (Questions, response options etc.) within reach at all times. This makes it very easy for the user to get an overview at any given time.

Secondly, a major part in providing a “content centered” experience is deciding what is not relevant. In the legacy application, a lot of features were crammed into a small view. This made the navigation cumbersome and did not highlight the most high priority features. QuestBuilder has a whole new approach to this. The features are prioritized and receive visual space according to their individual importance. This eases the use of the application and enables the user to engage in a far superior workflow.

"Will the correct action be sufficiently evident to the user?" [Spencer, 2000]

This is where the majority of the issues with the new web application arises. Often a feature might seem obvious for the experienced user and even the novice user. The problem lies in that the application lacks information for the first time user. The main view in figure 4.1 is a great example.

There is no clear indications as to where to begin. The correct action would be to drag one of the question types from the left pane into the center stage. This is obvious as soon as the user has done it for the first time. Although, for the first time user, there needs to be a more clear indication.

"Will the user notice that the correct action is available?" [Spencer, 2000]

The different actions within a certain view is generally visually accessible. As stated before, the available actions within every part of the application has been prioritized and given visual space accordingly. The result of this is an easy to navigate graphical user interface.

Even though the actions, in the form of buttons, icons, menu items etc., are generally easy to find the initial confusion that some views invoke for the first time user makes it hard to understand how to interact with the options at hand. As seen in the example earlier, the actual option for creating a question is clearly visible although how to interact is quite unintuitive and unclear.

“Will the user associate and interpret the response from the action correctly?" [Spencer, 2000]

This is where the new web application really shines. Every aspect of the application produces an appropriate response in the form of visual feedback. The user is always in control and has full awareness when it comes to what actions the software performs.

4.3. Heuristic Evaluation

The Questbuilder application is still in a development phase, and even though there is still much more to come, there is a lot of well developed and designed features that enables an efficient workflow. However, going with a minimalistic and more stripped down approach to design comes with a price. Some features become less intuitive for the first time user since the application overestimates the ability of the user.

Positive aspects

The new web application has adopted a content centered approach to design. This means that the produced content remains in focus throughout the application.

The goal with this approach is to enable the user to easily alter the provided content at any given time, making the application feel and work more efficiently towards the goal of the user, namely to construct a questionnaire.

Another aspect of the "content first" approach within Questbuilder is the way text is portrayed. The application has adopted a "What-You-See-Is-What-You-get" (WYSIWYG) design pattern. This means that the result of the input from the user is presented exactly as it will look in the final product, the actual survey. This might seem trivial but another, well adopted approach, is to provide one view for only editing and one for previewing the produced content. One example of this would be html-tags within text. The application would have one view where the user entered the text with the desired html-tags, then another view where it presents how the actual content would look.

The design also introduces a lot of areas where the application notifies the user of what task is performed. An example of this would be when a user adds questions from the database to the current survey. The application simply informs the user of what has happened and where to find the newly imported questions.

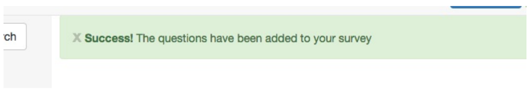

X **Success!** The questions have been added to your survey

Figure 4.8: Visual feedback provided by Questbuilder

Negative Aspects

The majority of what can be considered to be plain negative aspects of the new web application can be linked to how different actions or information is presented to the user. Questbuilder is developed with a very specific target audience in mind. This group of potential users have had previous experiences with the legacy application that is currently in use at AMA. These users grasps the basic technical vocabulary that is required to understand the application fully.

The downside of this is that a first time user, that has no previous experience with neither the new web application nor the legacy application, initially will have a hard time understanding some of the features. There is currently a very small amount of documentation, something that could solve this issue.

Questbuilder tries to eliminate the use for error messages. This is done by designing with error prevention in mind, i.e. not being able to insert numbers into a name field for example. Even though some areas of the application adopts this approach to the fullest extent, others needs to be improved to eliminate bugs and unintentional application behaviors.

The final negative aspect of the application can be seen as both a negative and a positive aspect. Questbuilder is a web application and with that comes a whole new approach to design that introduces new design patterns that does not exist in desktop applications, or at least are not as prominent.

This might inhibit some users from fully taking advantage of the application at first while adapting to these new principles. These are for example drag and drop that's very uncommon in legacy high-functionality applications such as the current desktop application.

But as the user starts getting used to this new way of providing a high-functionality application experience, the user will become more efficient and the project group is confident that the new web application enables the user to engage in an efficient workflow.

5. Development Phase

This section describes how the development of the web application QuestBuilder was carried out, problems that were faced during the development and how they were circumvented.

5.1. Modelling the surveys in Java

A big part of the development of QuestBuilder was the reverse engineering of the Survey file format. This is the format of the files that the original legacy software generated after creating a survey. The files consisted solely of numbers and boolean statements in addition to lines of plain text which were all divided by line breaks. This section will explain how the reverse engineering of these files was deducted and how the knowledge gained was used to create a data model of the files.

The back end, along with the data model, was written with the programming language Java. Java is an object oriented, class based language supporting concepts such as inheritance, polymorphism and encapsulation; all of which were important when modeling the survey files. Each question type has its own Java class representation and inherits attributes and functionality that is common for all question types from a parent class. Refer to 5.1.2 for an example of how Java's polymorphism has been used.

From a technical point of view, the purpose of QuestBuilder is to generate files in the Survey format that can be opened in the original legacy software. In order to manually create these files, some knowledge about their structure and how they are created becomes a prerequisite. The model that would come to act as the underlying structure of QuestBuilder would be created with this knowledge.

5.1.1. Analysis

An new empty file, created by the original legacy software, is the baseline for how Survey files are formatted. By comparing this file to a nearly identical file, but with some small changes made to it, the identification of what each and every line meant was determined.

By breaking down each and every type of element in the original legacy software this way, the understanding of how the elements were composed followed. The repeated task of changing a setting in the legacy software and comparing the file to its previous state, where that setting was different, proved to be useful.

As the analysis of the files went on, the requirement for a structured documentation grew. The analysis of the elements was kept in separate documents, one for each element. These documents contained the complete code for the entire element, with every line explained in detail. Keeping a well structured documentation was key in order to be able to make the analysis useful later on.

Another thing worth mentioning is also that these elements could be composed in a great deal of different ways. A typical element had more than ten parameters where these parameters could be affecting each other, which caused the analysis to prove more complex than first expected.

5.1.2. Result

From the analysis, a domain model was created. By understanding how the original legacy software functioned, a similar behaviour could be implemented in our model. Below is a diagram showing all classes that the model consists of, and how they are built upon each other.

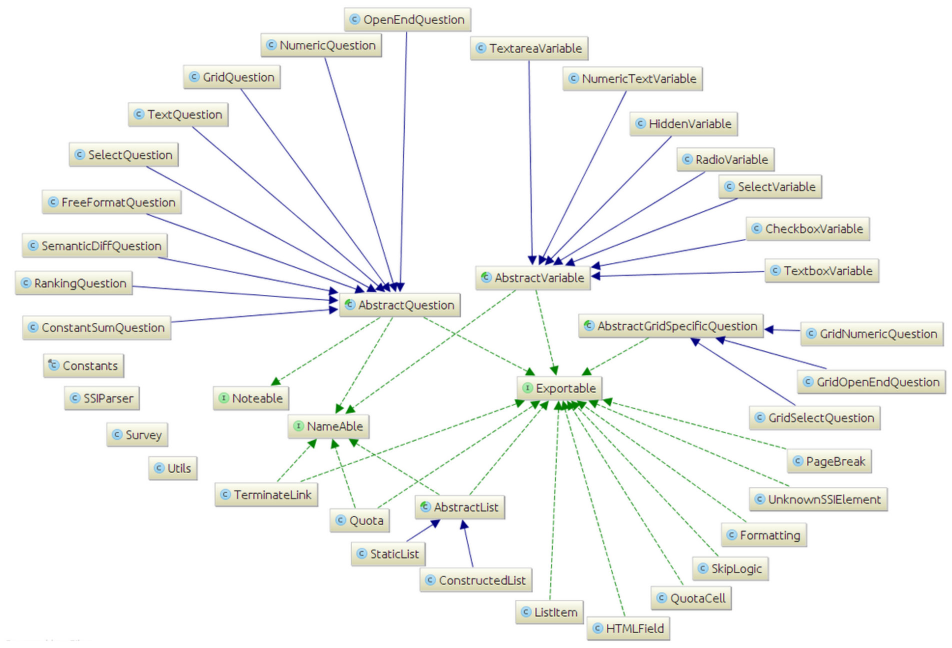


Figure 5.1: UML Diagram of the QuestBuilder model

The most important and interesting classes can be explained as follows:

Survey

The Survey object serves as a container for all other objects in a questionnaire. It contains lists of every exportable element such as questions, quotas, and lists. The Survey object, however, does not only work as a container for other elements. It also embodies some crucial core functionality, such as methods for exporting the survey to an Survey file that can later be opened in the original legacy software.

Methods for adding new exportable objects to the survey are also part of the Survey object and all these methods makes good use of the polymorphism that the Java programming language has to offer. When calling the “addExportable” method, the Java Virtual Machine (JVM) automatically decides which exact method to call based on what parameter that is passed to it.

Exportable

The Exportable interface is in some ways the core of the model. Every class that implements Exportable must also implement a way to export itself to the Survey file format. By having practically every used class to implement Exportable, the code to export an entire model to an Survey file becomes rather elegant: for every class in the model that implements the Exportable interface, run the export method. The rules for how each and every class should be exported is never the same, and the method itself is implemented differently in each of them.

AbstractQuestion

A Survey file is in general terms a Survey that consists of questions amongst other things. It turns out that these questions have many features in common, such as having a name or having some text to describe the question. In order to simplify the model the abstract class AbstractQuestion was created. Every type of question in QuestBuilder is an extension of this class and inherits the most basic parts that a question has. The specifics of each question is then implemented in the specific class. For example, the grid question makes use of a grid in order to store data whereas no other question type does this.

AbstractList

In addition to questions, a Survey has lists of response options. The grid in a grid question is made up by a row list where every response option represents a row, and also similarly a column list. The response options shown in a select question are also stored in a list.

There are two types of lists. Either a list is static or constructed. A static list is just a simple list with response options and there is nothing special about it. A constructed list, on the other hand, is a list where the response options shown may vary. The response options will be shown according to some logic such as “if this is true, then show that response option”. The response options to show originate from a parent list, which can in turn be another constructed list or a static list.

Since these list types are similar as a whole, but still vastly different in the details, it becomes natural to make use of yet another abstract class in order to make use of the object-oriented paradigm as much as possible.

AbstractGridSpecificQuestion

Apart from being a unique question type, the grid question can have three different response types. We refer to these as grid specific questions. For example, a response can be numeric, which means the user would be prompted to enter a numeric answer to a question. Just as with the other abstract classes, the grid specific questions have many things in common and are all extensions of the abstract class AbstractGridSpecificQuestion.

What makes this so useful is that the grid question must have at least one of these grid specific questions, but the model does not care for what type it is. Thus having a reference to the abstract class states that the grid specific question can be any of the three types.

SurveyParser

One important functionality of the backend is the ability to read Survey files and instantiate a Java object representing the data. This is the task of the SurveyParser. Given an input stream, it is possible to fully parse the file and create Java objects to represent the survey. An input stream is simply a continuous flow of data that becomes available over time and the stream can be connected to many different kinds of data sources, such as local files and network connections. The parser creates the Java objects needed to represent the survey by simply reading the data from the given input stream in the order it appears.

Thanks to the simple and well formed structure of the Survey file format, parsing files is rather uncomplicated. For example, each Survey element is designated by an identification number and this is extensively utilised when parsing data from a stream.

Each and every Survey element type requires its own method for parsing. When the parser encounters an integer that is known to be an identification number of a certain element, the appropriate method is called, which then handles the specific parsing of that element. After the method call is finished the parser simply continues with scanning the file for yet another identification number and repeats this process until there are no more data left to read.

It is not uncommon for a typical survey to contain elements that references each other in some way, and this turned out to be problematic when parsing a stream of data in the order it appears. For example, if the parser encounters data describing a constructed list whose parent list is yet to be parsed it is not possible to instantiate a java object representing the list until its parent has been parsed as well. This was solved by adding items with temporarily broken references to a queue and correct the references afterwards when the whole stream had been parsed.

5.1.3. Testing

The model was not tested until all the classes were written. The reason for this was that a passed test would mean that a file created with the model was compatible with the original legacy software. Just creating a separate instance of a certain question and exporting it to Survey format would thus not create a valid Survey file.

When the entire model was deemed ready to be tested, different Survey scenarios were simulated. A typical scenario for testing a specific question type was to create an empty Survey with just a single question of that type. The exported file would then be opened in the original legacy software and if it was able to read the file, the test was considered as passed. Different settings and parameters were tweaked and the test was repeated until every scenario for that question type had been tested.

As soon as all questions had passed the tests, surveys with different types of questions were created and tested in a similar way. Due to the fact that the number of different combinations that a survey could be composed in stretched over a thousandfold and that the testing process was time consuming, only the most common and reasonable combinations were tested.

Aside from this method of testing, a few bugs eventually appeared as QuestBuilder was developed. Whenever this happened it was due to small mistakes or a faulty detail in the analysis. These bugs were rather painless to solve and test in a similar manner as the model once identified, however identifying them was at times rather time consuming.

5.2. Building the front end

In a web application, the user usually does not notice what is happening in the back end. Even though the model is accurate and powerful, without a front end the back end is rendered useless. The front end represents the back end and is thus what the user sees and interacts with. It was written in JavaScript to be able to make use of tools such as AngularJS described in the section below. By using AngularJS it was possible to structure the front end according to the Model View Controller (MVC) pattern, making it modular, scalable and easy to organize.

5.2.1. Angular Modules

As previously mentioned in 2.1.3, AngularJS is one of the more popular JavaScript frameworks and was used when building the front end for QuestBuilder.

AngularJS makes use of modules, which is a key feature that separates the different parts of the application from each other. By making the application modular and delegate responsibilities amongst the different modules, the application embraces the scalability property that is required by a dynamic web application. QuestBuilder is made up by a separate module for each page displayed, along with a shared module. The shared module is used to handle the occasional need for passing data between separate modules.

Each module is typically made up by the three following parts:

Controllers
The controllers are mainly used to perform calculations and conditionally manipulate the DOM. Furthermore, controllers are typically also used to initiate needed data before a view is rendered. For example, a controller for a view meant to display all questions in a questionnaire might fetch the required data and set relevant variables before the view is ultimately displayed to the user.

Many of the controllers in QuestBuilder also includes methods to be called when the user performs a certain action in the graphical user interface, such as clicking on a button.

Services
The data fetched in the controllers is typically provided by a service. Services are only instantiated once, when the client opens a new instance of the web application, and all the components that depend on a service reference this single instance created by AngularJS. QuestBuilder makes use of a shared service in order to provide all the controllers with the same surveys. Due to the fact that the service is only instantiated once, if one controller modifies the survey the changes made will propagate to all other controllers as well.

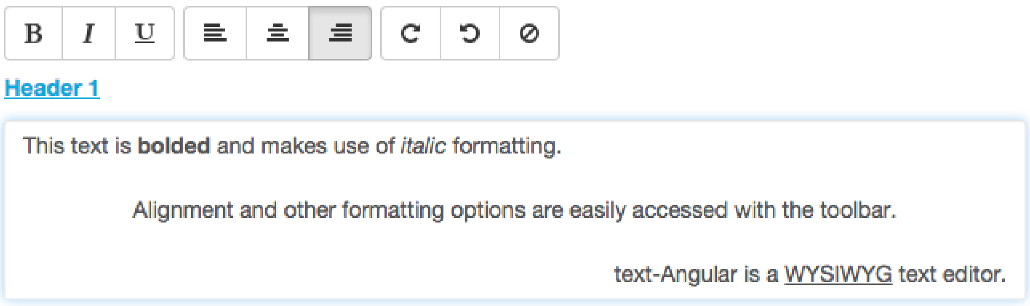
Views
This part contains all HTML files relevant to the particular angular module. Since the web application has been developed using the AngularJS framework many of the HTML files makes use of so called Angular directives.

An angular directive is a type of marker to be used on DOM elements that makes it possible to attach certain behaviour to them. For example, it is possible to specify when a DOM element should be visible based on a boolean expression and dynamically change the appearance of it. It is also possible to construct custom directives and specify custom DOM manipulation behavior.

5.2.2. Third party Angular modules

To be able to focus the development of QuestBuilder on functionality that is specific to this particular application, many third party Angular modules have been used. The AngularJS community offers an extensive list of Angular modules that can help extend web applications with commonly needed functionality and enhance visual appearances.

A suitable example of a third party module that is used extensively in QuestBuilder is “textAngular”. By including this module in the application it becomes trivial to create full fledged, customizable WYSIWYG text editor fields and use them for data input from the user. This module offers features such as keyboard shortcuts for applying bold font styling, undo/redo functionality in addition to other text formatting options as seen in figure 5.2 below.



This text is **bolded** and makes use of *italic* formatting.

Alignment and other formatting options are easily accessed with the toolbar.

text-Angular is a WYSIWYG text editor.

Figure 5.2: The text-Angular editor with a customised toolbar

Another third party module that has been of great significance during the development of QuestBuilder is “angular-drag-and-drop-lists”. It can be used to add drag and drop functionality to lists of almost any kind of DOM elements. The workflow of the so called “main view” of QuestBuilder, where the user is able to build a questionnaire, is built around this functionality. For example, to add another question to the questionnaire, the user simply grabs the wanted question type and releases it onto a list containing all current questionnaire elements as demonstrated in figure 5.3.

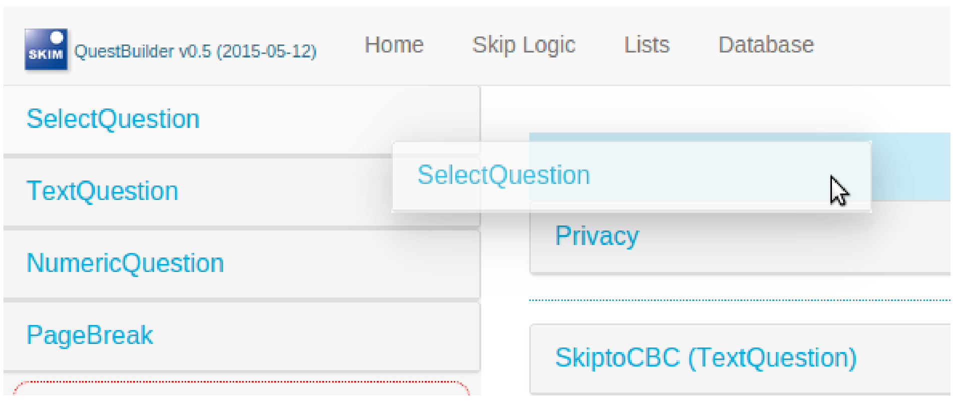


Figure 5.3: A question element being added to a questionnaire using drag and drop

5.3. Connecting the front and back end

In order to have a complete web application, communication is needed between the back and front end. The main communication consists of API calls from the client to the server, telling the backend to process survey files and generate Microsoft®Word compatible documents.

5.3.1. Converting between Java and JSON objects

As noted earlier, questionnaires are modeled as Java objects in the back end. In the front end, however, which is written in JavaScript, JSON objects are being used. Therefore, when passing data between the back and front end, a layer of conversion between the different object types is needed.

The conversion between Java and JSON objects is done solely in the back end using a Java library called Gson. This library is capable of easily converting back and forth between the two object types, often without any need for additional setup or configuration. However, some configuration was needed due to the use of abstract Java classes and interfaces in QuestBuilder.

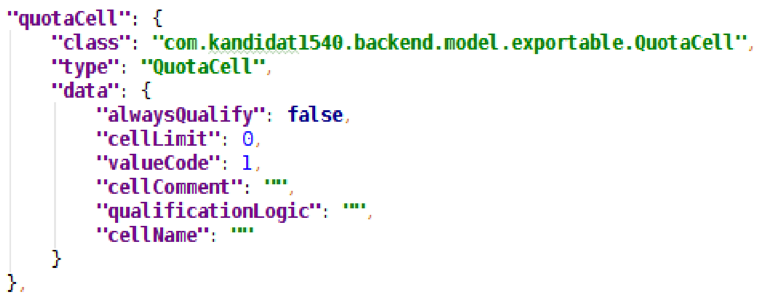
A Survey object, representing a questionnaire in Java, contains a list of elements in the questionnaire, all of which is of the type “Exportable”. The type declaration is as follows:

```
List<Exportable> ssiElements;
```

Figure 5.4: Declaration of the survey elements

However, each and every object in this list also has a sub type, such as “SelectQuestion”, “GridQuestion” etc that specifies the exact type of the element. Trying to convert a Survey object with such a list becomes problematic for the Gson library because it is not aware of the subtypes and therefore does not know how to convert the objects properly.

To solve this problem an adapter had to be created that, when converting from a Java object to a JSON object, appends a “type” attribute to the created JSON object specifying the actual subtype of the object. All other attributes originating from the Java object being converted is then wrapped in a nested “data” JSON object. Please refer to figure 5.5 for a text representation of an example object.



```
"quotaCell": {
    "class": "com.kandidat1540.backend.model.exportable.QuotaCell",
    "type": "QuotaCell",
    "data": {
        "alwaysQualify": false,
        "cellLimit": 0,
        "valueCode": 1,
        "cellComment": "",
        "qualificationLogic": "",
        "cellName": ""
    }
},
```

Figure 5.5: A JSON originating from a Java object representing a so called Quota Cell

With the adapter in place, the Gson library is able to convert from JSON to Java objects by using the “type” attribute for determining which type of Java object it should create.

5.3.2. Generating and sending files to the client

A web application does not have direct access to the file system of the client, which is both advantageous and disadvantageous. A direct advantage is the heightened level of protection by not allowing the application to access files on the hard drive. On the other hand, the complications that arise when the application actually needs to access files are clear.

QuestBuilder has the ability to generate Microsoft®Word compatible documents. The difficulty lies in how to transfer the generated document to the hard drive of the client that requested it. As mentioned in 5.1.2, a file sent to the server can be seen as an input stream. In a similar manner, a file sent to the client is an output stream. When the client requests a document, an output stream is opened that the back end can write the document to. When the stream is closed, the back end creates a link to the finished document that is automatically clicked and therefore requested by the client as a saved file. What the user sees is an indication that the server is producing a document and when the document is ready, a prompt to save the document is shown.

The process is very similar when the client requests a survey file. However, a survey file is represented with the Survey format whereas the Microsoft®Word compatible document uses the docx format. What differentiates these formats is how they are represented in pure text. A file using the Survey format can be opened and read by humans. A docx document on the other hand contains uninterpretable signs and is not readable by the human eye. In fact, creating and sending a survey file is more simple than sending a Microsoft®Word compatible document due to how they are encoded.

5.3.3. Persisting survey data

To be able to reuse often recurring standard questions such as "What is your gender?" and "Do you agree to this privacy statement?" QuestBuilder provides the possibility to persist sets of questions in a database whose resources are shared amongst all users of the application.

QuestBuilder has a MongoDB database and makes use of a library called Morphia for mapping Java Objects to and from MongoDB objects. All that is required to be able to persist a Java object using Morphia is to annotate the classes to persist and create simple data access objects that performs the CRUD (Create, Read, Update and Delete) operations on it.

The annotations used by Morphia is pretty straight forward. To make a Java class persistable the class declaration header is annotated with an "@Entity" tag. "@Id" tags can be used to specify which attribute of the class that should hold the database id and "@Embedded" is for telling morphia that an object referenced from within the class is to be embedded in the generated MongoDB object, instead of being persisted in a separate object.

6. Conclusion

6.1. Result

The QuestBuilder application offers a streamlined approach to constructing advanced surveys. It provides a minimalistic user experience where the user can easily access the most important features. This is the product of prioritizing features and giving them visual space based on their priority status. A complete visual review of the application can be read further in chapter 4.

A lot of the positive aspects of the new web application lies in the minimalistic design that makes it easy to navigate between views. By so doing it enables an intuitive workflow. Comparing to the legacy software, QuestBuilder empowers the users and gives him or her a better control over the process.

Even though the application is great in many areas, there are still issues that have to be addressed. Firstly, the application needs more documentation that facilitates the features to a first time user. The application assumes that the users have basic knowledge of the features implemented. Secondly, the application needs to provide a better error prevention strategy in every part of the application. Solving these issues will most definitely greatly increase the usability of the application.

As mentioned in section 3.2.1, AMA had ideas for how their new workflow should behave and look like. The main vision was to eliminate the repetitive tasks, in order deliver products of higher quality. To do this, the new workflow should contain a database where employees could share questions. The new workflow also included the dreams of an application that could generate both survey files and Microsoft® Word documents. This goal has been met by providing a far superior user experience than that of the legacy application. The prototype even provides the features that AMA wished for; QuestBuilder has a shared databased which makes it easier for employees to collaborate. The two consecutive steps of creating a Microsoft® Word document and to develop a questionnaire in the legacy application, can be done by only using QuestBuilder. After the questionnaire is developed in the prototype, it can be exported to a Microsoft® Word document. Since the survey file has been analyzed and modelled in the back end, a survey file can as well be exported. This makes it possible to add additional and advanced functionality and the legacy software afterwards.

The result is a software that will act as a complement to the legacy software, and not to replace it. This affected the results, since no additional features regarding the actual questionnaire could be added. No new question types could be added, nor additional behavior. The prototype therefore depended entirely on how the legacy application acted, and the survey file structure was the fundamental.

6.2. Discussion

QuestBuilder fulfills the required features within the limits of the project and has the capability of greatly improving the workflow of building questionnaires at AMA. Now, Microsoft® Word compatible documents can be generated automatically from a questionnaire, completely removing the need for creating such documents separately. Furhermore, sets of questions can be saved and then later on imported into new surveys.

The evaluation of QuestBuilder's GUI implies that it, in many ways, is superior to the legacy desktop application originally used by AMA. The more overview centered approach of QuestBuilder provides a constant awareness of how both the questionnaire and the document to be shared with the client will look like. QuestBuilder also has very few pop ups and windows obstructing the view of the application for the user compared to the legacy application. Thus, the navigation between the different views of QuestBuilder becomes more fluent and intuitive.

AMA intends to continue the development of QuestBuilder in order to cover even more features from the legacy software. The possibility to stop using the legacy software is however very low. If the goal was to eventually not use the legacy software, QuestBuilder would not need to be compatible with the file format of the legacy software and the limitations would be greatly reduced. However, the legacy software is required at AMA and QuestBuilder will greatly enhance their workflow.

The main focus of this project has been on delivering new features as fast as possible. Due to this, the planning and analysis phase was rushed, possibly resulting in design choices that were suboptimal. For example, the conversion back and forth between JSON and Java objects would not be necessary if both back- and front end had been written in JavaScript. Another downside of having such a focus on fast delivery of features is the lack of proper software testing. Without software testing it is hard to verify and prove the quality and correctness of QuestBuilder.

Due to the fact that a major part of this project was to provide a better user experience, evaluation of the GUI were of great importance. One could therefore argue that these activities could have been expanded upon and possibly involve potential users to a larger extent.

The project has proven to be a great learning experience for everyone involved. The authors have encountered many challenges that have resulted in a far greater understanding when it comes to both web application projects and large projects in general.

Not only the evaluation and the authors opinions matter in this case. AMA, the originators of the project and the company that will have an actual value of the end product, is pleased by the looks and functionality of QuestBuilder.

Bibliography

S. B. Barnes. Douglas carl engelbart: Developing the underlying concepts for contemporary computing. *Ieee Annals of the History of Computing*, 19: 16–26, 1997.

Andrew Beattie. Fyi on roi: A guide to calculating return on investment, 2014. URL `http://www.investopedia.com/articles/basics/10/guide-to-calculating-roi.asp`. Acquired 2015-05-18.

M. Beck, T. Beedle. Agile manifesto, 2001. URL `http://agilemanifesto.org/`. Acquired 2015-05-18.

Andy et. Al Brown. The uptake of web 2.0 technologies, and its impact on visually disabled users. *Universal Access in the Information Society*, 11: 185–199, 2012.

D.C. Chou. Cloud computing: A value creation model. *Computer Standards Interfaces*, 38:72–77, 2015.

Reuven Cohen. The cloud hits the mainstream: More than half of u.s. businesses now use cloud computing, 2013. URL `http://www.forbes.com/sites/reuvencohen/2013/04/16/the-cloud-hits-the-mainstream-more-than-half-of-u-s-businesses-now-use-clou` Acquired 2015-05-18.

A. Glore. P David. The impact of design and aesthetics on usability, credibility, and learning in an online environment, 2010. URL `http://www.westga.edu/~distance/ojdla/winter134/david_glore134.html`. Acquired 2015-05-27.

Vincent Driessen. A successful git branching model, 2010. URL `http://nvie.com/posts/a-successful-git-branching-model/`. Acquired 2015-05-18.

T. Dyba, T. Dingsoyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50:833–859, 2008.

G Fischer. User modeling in human-computer interaction. *User Modeling and User-Adapted Interaction*, 11:65–86, 2001.

Joe Gebbia. How design thinking transformed airbnb from failing startup to billion-dollar business, 2013. URL `https://www.youtube.com/watch?v=RUEjYswwWPY`. Acquired 2015-05-18.

GoogleTrends. Compare angular js, backbone js, ember js, knockout js, react js, 2015. URL `http://bit.ly/1PQqnzn`. Acquired 2015-05-18.

The Open Group. Cloud computing for business : What is cloud?, 2011. URL `http://www.opengroup.org/cloud/cloud/cloud_for_business/what.htm`. Acquired 2015-05-18.

N. Hazzenzahl, M.; Tractinsky. User experience - a research agenda. *Behaviour Information Technology*, 25:91–97, 2006.

ISTQB. Why is software testing necessary?, 2015. URL http://istqbexamcertification.com/why-is-testing-necessary/. Acquired 2015-05-18.

PRAKASH JOTHIRAMALINGAM. Desktop application development : Is there a future?, 2012. URL https://devmanagement.wordpress.com/2012/08/22/desktop-application-development-is-there-a-future/. Acquired 2015-05-18.

J. Kumar. Humanizing the enterprise. *Design, User Experience, and Usability. User Experience Design Practice*, 25:61–70, 2014.

Daniel Kummer. git-flow cheatsheet. URL http://danielkummer.github.io/git-flow-cheatsheet/. Acquired 2015-05-18.

Willy Lai. User experience design for roi, 2014. URL http://uxstrategiessummit.com/assets/files/pres/UXSSSanFrancisco_WillyLai_upload.pdf. Acquired 2015-05-18.

G. Dudekx C. Brown J. Lindgaard, G. Fernandes. Attention web designers: You have 50 milliseconds to make a good first impression! *Behaviour Information Technology*, 25:115–126, 2006.

Jakob Nielsen. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 152–158, Boston, Massachusetts, USA, 1994. ACM.

Jakob Nielsen. How to conduct a heuristic evaluation, 1995a. URL http://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/. Acquired 2015-05-18.

Jakob Nielsen. 10 usability heuristics for user interface design, 1995b. URL http://www.nngroup.com/articles/ten-usability-heuristics/. Acquired 2015-05-18.

Jakob Nielsen. Usability roi declining, but still strong, 2008. URL http://www.nngroup.com/articles/usability-roi-declining-but-still-strong/. Acquired 2015-05-18.

Jakob Nielsen. The magnifying-glass icon in search design: Pros and cons, 2014. URL http://www.nngroup.com/articles/magnifying-glass-icon/. Acquired 2015-06-01.

Tim O'Reilly. "what is web 2.0: Design patterns and business models for the next generation of software. *International Journal of Digital Economics*, 65: 17–37, 2007.

H. Preece J. Rogers, Y. Sharp. *Interaction Design: Beyond Human - Computer Interaction.* John Wiley Sons Ltd, 3 edition, April 2011. Chapter: 15.2.2.

J. Schwaber, K. Sutherland. The scrum guide™, the definitive guide to scrum:the rules of the game, 2013. URL `http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf`. Acquired 2015-05-18.

Rick Spencer. The streamlined cognitive walkthrough method, working around social constraints encountered in a software development company. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 353–359, The Hague, The Netherlands, 2000. ACM.

Frank Spillers. Making a strong business case for the roi of ux, 2014. URL `https://www.experiencedynamics.com/blog/2014/07/making-strong-business-case-roi-ux-infographic`. Acquired 2015-05-18.

J Tidwell. Center stage, 2015. URL `http://designinginterfaces.com/firstedition/index.php?page=Center_Stage`. Acquired 2015-05-17.

A. Appendix

A.1. Contributions

The planning of the project was done continuously throughout the project, since Scrum was chosen to be the project method. The sprint duration was two weeks, and at the end of every sprint AMA reviewed the work and the group received feedback.

A.1.1. Responsibilities

Filip Carlén

Analysis of Legacy Software (GUI)
Front End structure
GUI Research
GUI Drafts
Main View
Select Settings
Database Export
Analysis of New Software

Hampus Forsvall

Analysis of Legacy Software (GUI)
Front End structure
GUI Research
GUI Drafts
Main View
Shared Database
Database View
Analysis of New Software

Simon Persson

Analysis of Legacy Software (survey files)
GUI Drafts
Model
Back end
Server setup
Front and back end communication
Import and export of survey files
Main View
Skip Logic View
Back end refractoring
Back end testing
Visual authentication
Persistence

Erik Tholén

Analysis of Legacy Software (survey files)
GUI Drafts

Model
Back end
Server setup
Front and back end communication
Import and export of survey files
Main View
Skip Logic View
Generation of Microsoft® Word compatible documents

A.1.2. Problem solving, synthesis, and analysis

Problems that encountered members of the project, were more often solved individually or in pairs. Major issues that influenced the project as a whole were discussed at a group meeting until an agreement had been reached.

A.1.3. Report contributions

Filip Carlén
Sections: Purpose, Problem, 1.5.1, 2.2.1, 2.2.2, 2.2.4, 3.1-3.3, 3.4.4, 4.1, Result, Discussion

Hampus Forsvall
Sections: Problem, Limitations, 2.3, 3.4.1-3.4.3, 4.2, 4.3, Results, Discussion

Simon Persson
Sections: Abstract, Sammanfattning, Introduction, Acknowledgements, 2.1, 5, 6

Erik Tholén
Sections: Abstract, Sammanfattning, Introduction, Acknowledgements, 1.2, 1.4, 2.1, 5, 6

A.2. Interview with AMA

How does the creation of surveys looks like at AMA today?

“Today the creation of surveys contains two steps. At first, the employee needs to produce a text document in an optional word processor. These text documents are styled according a company standard. The meaning of these documents is to give our clients a brief and easy overview of what the survey will contain. Another meaning of the text documents is that the clients can have the survey documented. All questions that will be implemented in the survey are listed together with response options and comments on how the skip-logic will work. When this step is considered to be done, it will be sent to the client for feedback. The feedback will be taken into account, and the text documents are changed according to the feedback.

The second phase is the creation of the actual survey. We use a software for making these surveys. The survey will look and behave exactly to the agreement that have been met before this step. Even though an agreement has been met, the survey will receive another round of feedback before it’s time for the survey to go live!”

What are the major issues with the current workflow?

“The major issue with the current workflow is all of the repetitive tasks that it contains. When creating the text document, we usually tries to find questions that have been used in earlier questionnaires. These question do we paste into the new one. This is extremely inefficient. And when we are trying to implement the survey in there are no possibilities to import old questions. Therefore we have to do everything over and over again. Since the current software does not give any pleasant experience, the employees at AMA usually gets frustrated.

Meeting clients’ demand of faster delivery, requires efficiency in all phases of the internal project process. Therefore we need to make the current workflow much more efficient.”

What are the major issues with the current software?

“There are two fundamental issues with the current software. At first, it does not give the users a pleasant experience at all. It is not intuitive and easy to use. Even though our employees have learnt how to use the features, it would still be better if all the features not used on a daily basis, could be hidden in order to make the important ones much more easy to use. It is also difficult to have a total overview of what’s going on, and what has been done previously. It would be much better if you could see how the question would look like by default, instead of pressing preview buttons.

The other fundamental issue is the lack of possibilities to reuse old questions. It can be done, but it's not smooth at all. An easy way to save frequent questions would be a feature with a high value for our company."

What are your visions for how a better workflow would look like?

"The overall objective is to provide a software that makes the creating process of clients' questionnaire and surveys more efficient, standardized and of higher quality. One ambition is to merge the creation of text documents and surveys into one process, instead of two. To gather these two separate processes into a hub where both the text document and the survey file can be exported from. In this hub, it would be easy to overview how the survey would look like in the end. If the styling of the question types in the survey could look like the styling in the text document, that would be fantastic. The new software, should be good looking and easy to use. All features that are not used on a daily basis should be hidden, and the more commonly used ones should be very easy to use.

What is very important is to have a database where all of employees could save questions for future use. For example, questions regarding age, gender and home district are used in almost every survey. They should be saved in a set which can be easily imported in another survey. Since the database is shared, a large amount of questions could be easy accessible. If this works, our company would save enormous amounts of time."

Which benefits would AMA gain from a new workflow?

"A more efficient process will result in higher quality, cost savings and more available time for fun and value-added work. Cost savings will gain efficiency and result in higher project profitability. A more standardized process would result in higher and more consistent quality. Reducing repetitive and dull tasks and doing more stimulating work will increase employees' satisfaction."

Should the current application be replaced, or do you still want to use that software to make changes and adding more advanced features?

"Of course, it would be fantastic to replace our current software by 100 percent. But since there are an enormous amount of features implemented, it feels like an impossible assignment to replace it all. The functionality the current software offers, regarding exercises, works fairly good at the moment. It is also a very advanced feature, which is almost impossible to imitate in another software. If we could have a software that does everything that we are asking for, such as database import/export, text document generation and gives a much better user experience, it would be enough to start with. New features can be implemented successive. We can use SSI Web to complement the questionnaires exported from the new software. Same with the text document generation. We can complement it. We still believe that the amount of time that will be saved by doing it this way, have a powerful impact for us as a company."

A.3. GUI Evaluations

Tasks

1. Create survey
2. Create a Select Question
    a. Name: S1
    b. Header 1:“Are you over 18?
    c. Response Options: “Yes/No”
    d. Set Randomize
    e. Set respondent specify on option “No”
    f. Activate none of the above on response option “Yes”
    g. Remove response option “Yes”
3. Skip Logic.
    a. Question S1
    b. Alternative under 18
    c. Terminate
4. Create a constructed list based on if you are over 18. The constructed list will then be used on the next question to determine the last response option.
    a. “ Tea/Coffee/Soda/(If the respondant is over 18, then beer, otherwise juice)

Legacy Application

Cognitive Walkthrough

1. Creating a survey
    a. File -> New Study
        i. Yes, this follows the standard desktop application design principles. Although the other “New Survey” button is unintuitive and redundant.
        ii. Yes
        iii. Yes, the pop up makes it very clear
    b. A pop up with different input fields appears
        i. Yes
        ii. Yes
        iii. No, the pop up disappears but workspace behind it remains the same. There is very minor change (A small text changes to the name of the survey)
2. Create a Select Question
    a. Create Questionnaire by Clicking the “Pencil icon”
        i. No, the icon does not conform to the standard. That a survey contains a Questionnaire is quite unintuitive as well. Also there are many ways to open the questionnaire window rather than having one clear choice.
        ii. No, there are plenty of buttons but none that clearly show which one to choose.

        iii. Yes, it is quite apparent what the menu is for.
    b. Name the question “S1”
        i. Yes, the name input field is easily visible
        ii. Yes
        iii. Yes, the user arrives at the select question edit page
    c. Add Header 1: “Are you over 18?
        i. Yes, the text field is visible
        ii. Yes
        iii. No, even though the text appears in the input field there is no way of knowing how it will appear in the actual survey.
    d. Add Response options “Yes/No”
        i. Press the Response Options tab
            1. Yes, the tab is clearly visible
            2. Yes
            3. Yes, the “Add” button is easily visible and, in conjunction with the empty text pane, indicates that this is where the response options should be entered
        ii. Press the Add button
            1. Yes, the input field that appears indicates that this is where the response options should be typed.
            2. Yes, it appears in the middle of the screen
            3. Yes, the response options appears in the initial text pane when you press “ok”.
    e. Randomize Respone Options
        i. Tick the checkbox
            1. Yes, the checkbox is very visible
            2. Yes
            3. Yes, even though there are no feedback.
    f. Respondent specify on “No”
        i. No, the option is greyed out as long as no response option is selected
        ii. Yes, the option is clearly visible
        iii. Yes, this is similar to the randomize option in this regard
    g. Activate none of the above on response option “Yes”
        i. No, the option is greyed out as long as no response option is selected
        ii. Yes, the option is clearly visible
        iii. Yes, this is similar to the randomize option in this regard
    h. Remove response option “Yes”
        i. No, the button does not conform to the standard
        ii. No, even though the button is visible, the buttons next to it look as if they could contain a delete button rather than having one with “Delete” in plain text
        iii. Yes, the option disappears from the list
3. Skip logic
    a. Press skip logic

        i. No, the skip logic menu is not linked to the actual question
        ii. Yes, the button is very visible
        iii. Yes, the pop up indicates that this is the skip logic view
    b. Press Add
        i. Yes, everything is greyed out to indicate that a new object has to be created
        ii. Yes, the add button is quite visible
        iii. Yes, the greyed area is no longer greyed out and the skip logic name is shown in the list
    c. Choose S1 in the "Skip from" dropdown
        i. Yes
        ii. Yes
        iii. Yes
    d. Type the code for the the "No" response option
        i. No, this is very unintuitive and requires training
        ii. No, there is no real indication as to where the user should insert the code
        iii. No, there is no feedback that indicates that the code is correct
    e. Choose terminate in the Skip to dropdown
        i. Yes
        ii. Yes
        iii. Yes
4. Create a constructed list
    a. Open List menu
        i. No, the icon is hard to see and understand
        ii. Yes, it is easily visible
        iii. Yes, the pop up window indicates that this is the list view
    b. Press Add constructed list
        i. Yes
        ii. Yes
        iii. Yes
    c. Choose parent list
        i. Yes
        ii. Yes
        iii. Yes

Heuristic Evaluation

1. Create survey
    a. The user can easily follow along since, when the "new survey" button is pressed, a new pop up appears.
    b. The file path should be hidden, it's irrelevant.
    c. The new survey icon does not follow the uniform standard.
    d. Does not prevent errors, display error messages instead.
2. Surveys' been created

   a. the overhead view doesn’t really provide any information and can be confusing.
   b. “Data fields” is not a good word for survey items.
   c. The application provides a lot of features on the menu bar, should probably hide some?
   d. The language is a bit too technical
   e. Icons are not uniform to the standard metaphors.
   f. A lot of empty space, does not take advantage of the workspace.
3. Create Question
   a. “Write Questionnaire” is not a good way to communicate the action.
   b. The “Write Questionnaire” view is the main view that the user interacts with and therefore it should be the first view to appear after creating a survey.
   c. The Program allows for cancellation of the user's action, in the form of “Cancel” and “Close” buttons.
   d. The save button doesn’t give any feedback as to what it actually does.
   e. No good overview of the questionnaire, you have to actually open up the individual questions to get an idea of what that say.
   f. Again, data fields…
   g. Icons aren’t descriptive of what actions they perform(magnifying glass isn’t search…)
   h. It’s a very messy interface
4. Add to Questionnaire view
   a. abbreviations are unnecessary and confusing
   b. Requires the user to understand every question type in advance
   c. Doesn’t prevent errors
5. Select Question view (question text)
   a. Requires the users to know what a question contains in SSI (Header1, Header2 etc..)
   b. No overview, needs to press preview to see changes
   c. Pen and pencil buttons are unnecessary, should provide these functions in the actual view
   d. Error when previewing an empty question is fairly good, allows the user to fix them
   e. Why is rename a separate button and view?
6. Select Question view (response option)
   a. Alot of dead space
   b. What is an existing list and should I make a new list? The existing list is actually the list for this question, so why would I create a new one?
   c. Add button is easy to find but why does it open a new view?
   d. “Are you sure” when deleting a response option, not that big of a deal
   e. The language is very technical and could easily confuse a novice user
   f. Non-minimalistic, displays all the features at once
   g. Different names for the same thing throughout the view
7. Select Question view (Settings)
   a. This might be the best view in SSI
   b. Only displays relevant information

    c. Clarifies what the different select formats means
    d. Some features might be redundant though but overall a good view
8. Skip Logic
    a. The skip logic option is easy to find
    b. Half the view is greyed out if the user hasn't created a skip logic yet
    c. Delete displays error messages if no SL is chosen
    d. Why do you need to press a button to validate your code? should be done automatically.
    e. Agian, icons are all messed up
    f. Print allows for printing empty skip logics
    g. Text editor is redundant
    h. The skip logics are not visible from the main view which could make debugging a hassle
9. Constructed list
    a. Displays a "help text" that says "To edit a list, select a single list on the left", This is greyed out though, should be more visible
    b. The list allows for multiple select even though it doesn't support it properly
    c. When selecting a list, the view displays the same views as when adding response options to a question. This is fairly good since the user recognizes the interface
    d. Constructed lists uses a very confusing PERL scripting interface. Although it gives no indication as to how this should be performed. You have to know what to do otherwise you're lost. The help function shows a list of API calls, this doesn't really help the novice user at all.

Questbuilder Evaluation

Cognitive Walkthrough

1. Create Survey
    a. It's unclear if you've already made a survey or if you need to create one somehow
    b. Same here since you're not sure how to proceed if you don't know whether or not you've got a new survey
    c. -
2. Create SQ
    a. The option might be unclear if the user is not familiar with drag and drop
    b. The option is in plain sight
    c. The question is shown in the center stage
3. Name the question
    a. It's very clear
    b. Should probably add a "ok" button
    c. Yes!
4. Set header 1 to "Are you over 18?"
    a. It's very easy

    b. The input field is in plain sight
    c. Yes
5. Response options
    a. Yes, you can easily see the response options
    b. It’s very easy as it’s in plain sight
    c. The options are displayed in plain sight
6. Set randomize
    a. It’s not that clear at first, the user must press the settings button
    b. When the user has entered the settings view the option is clearly visible
    c. The checkbox displays that the option has been chosen.
7. Set Respondents specify
    a. No, the option isn’t that clear
    b. The button is clearly visible but it’s hard to tell what it does
    c. Which response option that’s chosen is hard to see.
8. Activate non of the above
    a. Same as the previous task
9. Remove response option
    a. Yes the option is very clear in the settings view
    b. Yes
    c. Yes, the option disappears
10. Skip logic, when not over 18 terminate
    a. Fairly, the text describes how proceed
    b. The button/list item is very visible
    c. The user shifts his attention to the middle, making it less obvious that he needs to pick an option on the left
11. Constructed List
    a. Yes, the info pane describes what has to be done
    b. Yes
    c. Yes, there’s a save button

Heuristic Evaluation

1. Create Survey
    a. It’s not quite clear whether a survey is created from the start or not
    b. It’s unclear where to start
    c. The application does not use a language that is **too** technical since a certain level of knowledge can be expected
    d. Minimalistic design
2. Select Question
    a. Drag and drop needs to more clear
    b. When a question is created it’s very easy to get an overview of the application
    c. It’s unclear which fields that has to be filled out to proceed
    d. The view is not very descriptive, requires previous knowledge
    e. After the question has been added, the view provides great overview

    f. The edit button doesn’t really show whether or not the user is in the “edit mode”. Needs to be more clear
    g. The view could use more tooltips to describe the different components
    h. It’s not obvious that questions can be rearranged using drag’n’drop

3. Select Question (Settings)
    a. Minimalistic design
    b. Easy overview
    c. The button to add lists is a bit unclear
    d. It’s hard to tell which response options are “NotA” and Respondent Specify
    e. The question name field is a bit unclear, needs some sort of label
    f. The input fields includes HTML
    g. More tooltips to describe the different aspects
    h. Checkboxes are not a great way to present togglable features
4. Skip Logic
    a. The information is clear but needs to adapt to the amount of questions at hand
    b. The text that states the sort of question is too small
    c. The mousepointer is the same as the one in the main view that indicates that an item can be moved using drag’n’drop even though it isn’t available
    d. The questions need to, more clearly, indicate how they are selected
    e. The mouse pointer changes when interacting with the left panel, even though it doesn’t change the behaviour
    f. The language is still a bit technical, needs some sort of tooltip to clearify
    g. “Clear FROM question” button is not very clear, needs to be rephrased
    h. The left side should indicate more clearly how many skip logics that falls under the question
    i. What does the AND/OR buttons at the bottom right mean?
5. Constructed List
    a. The language is quite technical, needs reworking and clarification
    b. The drag’n’drop feature is not clearly presented, needs some sort of visual queue
    c. When dragging, it’s hard to know where you can drop the item
    d. The options shouldn’t be selected by default
    e. Needs tooltips to ease the learning curve of a novice user
    f. Great overview!
    g. The different components aren’t labeled, which makes it hard to distinguish the different features