Job Aid for Neonatal Nurses

Jessica Chayavichitsilp
Computer Science & Engineering
185 Stevens Way
Seattle, Washington 98195
chayaj@uw.edu

Tobias Kahan
Computer Science & Engineering
185 Stevens Way
Seattle, Washington 98195
tobyk100@uw.edu

Xiaoxia Jian
Computer Science & Engineering
185 Stevens Way
Seattle, Washington 98195
xiaoxiaj@uw.edu

ABSTRACT

Nurses working in Neonatal Intensive Care Units (NICUs) perform complex procedures and assessments and turn to a variety of resources to aid them in their daily work. The resources they turn to are often desktop applications or web searches, causing the nurses to return to their station to look for information. To complicate matters, many of the procedures and diagnostics are based on an infant's weight and so the nurses must remember or look up various formulas throughout the day. Fortunately, more and more nurses carry smartphones and we intend to leverage this technology, already in the hands of nurses, to provide the information and computing power they need.

Mobile apps for neonatal nurses already exist, as a search for "neonatal" in the Apple App Store will indicate. However, we have found that none of these apps contain the right blend of information for neonatal nurses. Some have too much information, which makes navigating the app difficult, and some are specific to a single procedure, which causes the nurse to flip through multiple apps to get all of the information they need. We have designed an application in close coordination with a team of neonatal nurses from the University of Washington, which provides information on the most frequently used procedures. Our final evaluation will be if our app can make the job of neonatal nurses easier.

1. INTRODUCTION

Neonatology is a subspecialty of pediatrics that consists of the medical care of newborn infants, especially ill or premature newborn infants. Neonatal nurses work with very tiny patients that require special care, performing procedures quite differently than when done on older and larger patients. To account for the unique procedures performed by neonatal nurses, NICUs have developed their own internal resources. Along with these internal resources, generally desktop applications, neonatal nurses turn to Internet searches for information. There are various problems with these resources and a group of nurses at the University of Washington Medical Center has asked us to design an application that will solve the limitations of the existing technology.

To understand the problems, which exist in the current solutions, we can imagine a nurse caring for their patient. The nurse is about to perform a procedure, such as administering medication, but they forget the exact ratio of medicine to weight. To figure out this information they turn to their internal neonatal care website. At this point they encounter two problems; first, the internal website is desktop based and so the nurse must return to the nursing station before continuing with patient care. Second, the abundance of information on the site impedes the nurse from finding the most commonly used material; the site is cluttered. Either the nurse wastes valuable time returning to their station or they choose to avoid using the website and instead they rely on memory to perform the procedure. Often times, a nurse remembers the correct dosage but they may want to confirm it as an extra precaution. In this case the time spent confirming their recollection might not be worth it and they proceed with administering the medicine. We want to help in both cases, providing information that nurses have completely forgotten as well as quick refreshers for something the nurse wants to confirm.

As stated above the existing solutions are desktop based and cluttered; we intend to solve both of these problems. To solve the first problem, we will create an app optimized for smartphone devices, allowing the nurse to seamlessly continue with patient care. To solve the second problem, we will include only the most used procedures, keeping the app easy to navigate. However, our solution immediately runs into its own problems. First, the smartphones carried by nurses run on different operating systems and thus we must design a solution that is cross-platform compatible. Second, how do we know which procedures to include and how do we organize the app to make accessing this information painless for the nurses? We will tackle each of the issues in turn.

To make our app cross-platform compatible we will use the Open-Data-Kit (ODK) framework developed by the Computer Science department at The University of Washington. The ODK platform lets developers create an ODK Survey within a spreadsheet, which is transformed into a working mobile app. This mobile app is built on top of jQuery Mobile, a platform that allows developers to write a JavaScript app and have it render and function consistently on a wide range of platforms, including the two most popular mobile platforms: iOS and Android. There are drawbacks to using jQuery Mobile, it feels slightly less native and is a good deal slower than native apps. However the opportunity to write one app and have it work on multiple devices outweighs this drawback. There are also drawbacks to using ODK, it doesn't support everything we need in our app. Fortunately, ODK allows you to edit the HTML and JavaScript underneath the hood, so it is really as extensible as you want.

In order to decide which features to include in our app we coordinated with Jocelyn Kirk, a neonatal nurse working at the University of Washington. Beyond that, we tested our design with a group of different nurses and worked their feedback into a new design. This is discussed in the evaluation section.

The remainder of this paper is organized as follows. Section 2 discusses related work and how our application differs from those on the market. In Section 3 we examine the approach – how we designed the app. Section 4 convers the implementation and in Section 5 we discuss how we plan to evaluate our work.

2. RELATED WORK

The ground is set for mobile technology to revolutionize the nursing field. In fact, the hardware is already in the right hands: seventy-nine percent of health care professionals report using a smartphone in clinical practice [1]. Seventy-seven percent of current students in medical professions own a smartphone and a similar percentage of those that own smartphones already use medical apps [2]. The two most common uses of medical smartphone apps are medication guides and discipline specific guides [2]. A search of the iTunes app store reveals over 1000 medically related apps, and the search 'NICU' (Neonatal Intensive Care Unit) returns 30 hits. Among these NICU results there are two general categories of app, procedure specific apps and general neonatal reference apps. One such specific app is *NeoTube,* which allows the user to enter a baby's weight and outputs measurements and dosages specific to that weight [3]. Another procedure specific app is *Multidrip Glucose Infusion Rate Calculator,* which, like *NeoTube,* starts with the nurse entering the baby's weight and outputs the dextrose percentage and rate of infusion for the five drips, which constitute the treatment [4]. A benefit of these procedure specific apps is that their interface is simple and they can provide detailed information on the procedure. Additionally, a nurse can compose a unique set of procedure specific apps. Of course, the downside is that a nurse has to switch through many apps in the course of her or his work. The benefit of a general neonatal reference app is that a nurse sees a hierarchical display of the resources and can even enter the baby's weight once and have that propagate to multiple calculations. One general app is *Neonatal Calculator,* which lets the user enter the baby's weight and outputs drug dosages, infusion rates, apgar score, and numerous other calculations [3]. This app is extremely useful and popular however it doesn't include any reference material on how to perform the procedures. *SimNICU* provides a wealth of reference material and calculators [7] as does *Pedi STAT* [8], however both apps are missing certain common procedures and resources. Our app follows in the footsteps of these established apps, however, our design is based on the experience and feedback of multiple nurses currently working in neonatal units. We believe that our close collaboration with experienced neonatal nurses will generate an app that contains the most utilized resources while omitting less popular information that would clutter the interface. We have included a comparative list of features, which shows how our app is different from what is out there (See: Appendix 10)

3. APPROACH

As stated in the introduction, we are creating a mobile application to provide to neonatal nurses the most commonly sought information and most commonly used calculations in a mobile friendly format. We did not fully complete the application, however, we provided examples of three categories of resources: medications, diagnostics, and newborn tools. We will detail those categories as well as some design decisions we made.

Nurses deliver medication based on the weight of their patient. Thus, for each medication the nurse must reference a unique formula. To simplify this process we encapsulated the formula for each medicine and provided the nurses with a single tool that accepts a medication and the baby's weight and outputs the dosage. We only implemented three example medications; see figure 1 and 2 for how this tool works.

The next example category we implemented is diagnostics. When diagnosing an infant for a chest infection, neonatal nurses must consult reference x-rays, which they compare to those of the patient. Again, this information is found at the nursing station and so we put this material in our app. Figure 3 shows how we interspersed text and images to give the nurses a quick reference on this material.

Finally, we provide an example of a newborn assessment: the Ballard Exam. The Ballard Exam consists of 14 subcategories that can be completed in any order. The scores from each category are summed into a total score, which reflects the developmental stage of the infant. Figure 4 shows the appearance of the Ballard Exam.

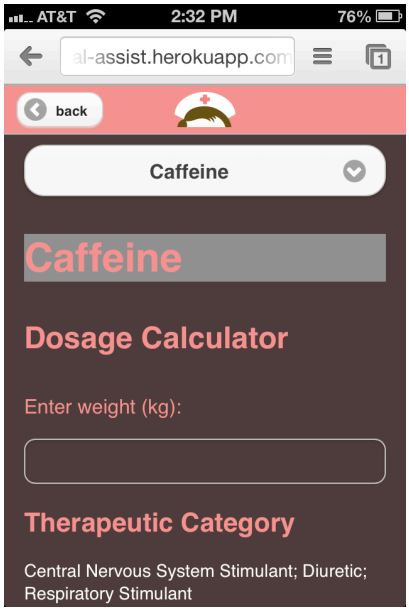


Figure 1: A page where the nurse can select a medication and enter the baby's weight. The proper dosage is then emitted on the screen.

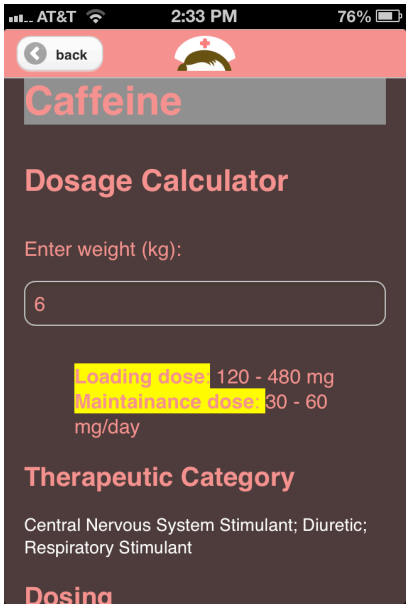


Figure 2: The screen after the nurse has entered a weight.

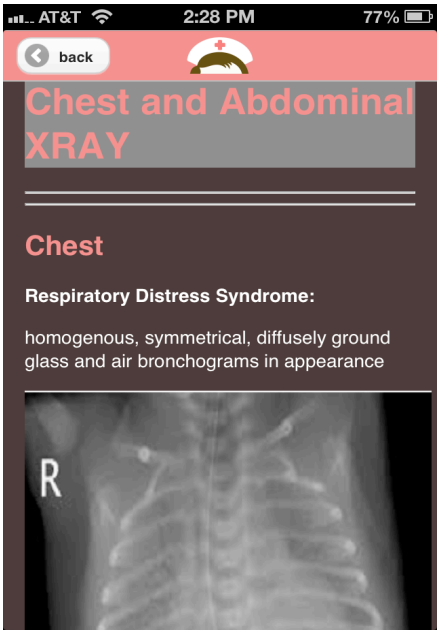


Figure 3: Reference x-rays for chest and abdomen.

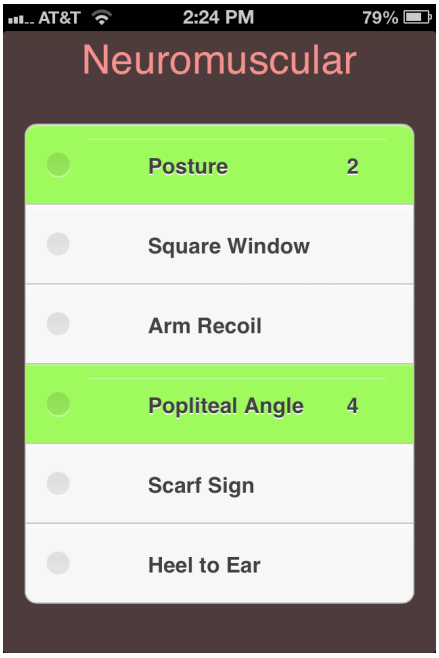


Figure 4: The Ballard Exam.

Besides providing this content we had to decide how to organize it. We decided to use a framework called Information Architecture (IA), which is a hierarchical representation of the application's various screens. The initial design of the IA consisted of a main menu with nine categories. Our alpha testers responded that the main menu was too cluttered and so we reduced the items on the main menu. This resulted in deeper paths from main menu to desired content, meaning that the nurse has to navigate through more screens to get the desired content. However, the main menu is simpler and easier to read which hopefully speeds up the time from opening the app to finding the desired content. We have included a diagram of the information architecture in Figures 5 and 6 showing the old IA, which is cluttered, and the new IA, which has more levels but less options on the main screen (the one showed in green). Please refer to Appendix 11 for full size versions of the Information Architecture. Another design decision to note in figures 5 and 6 is the duplicate paths to certain resources. In figure 6 we have included dotted lines which connect a resource to its' placement in another part of the hierarchy. For instance, the nurse can find the weight loss calculator in either Nutrition Tools or Calculators. This has the downside of adding more total buttons to the app and possibly causing the confusion of mistakenly thinking the two calculators are different. However, the benefit is that nurses who forget where a certain item is located will be able to find it more easily. For instance, most of our calculators fall under some specific category but we have also listed them under "Calculators" which makes any calculator easy to find.

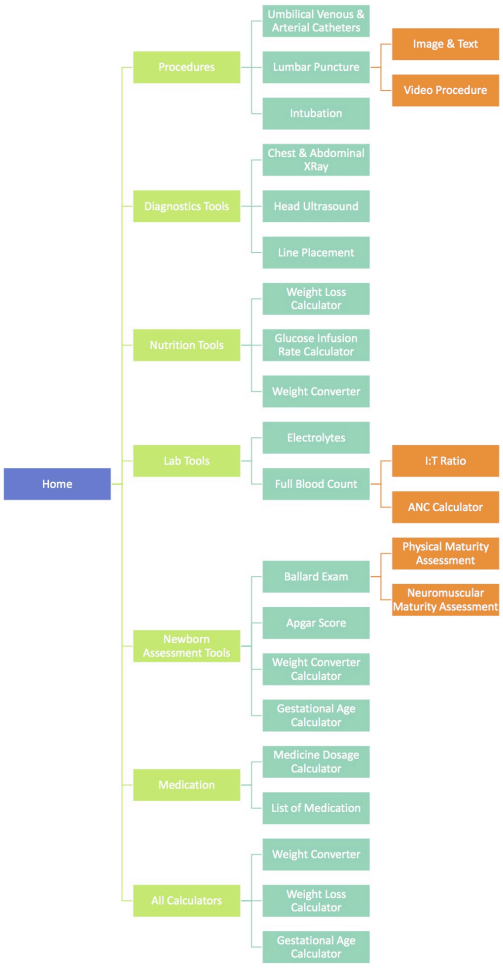


Figure 5: The old Information Architecture.

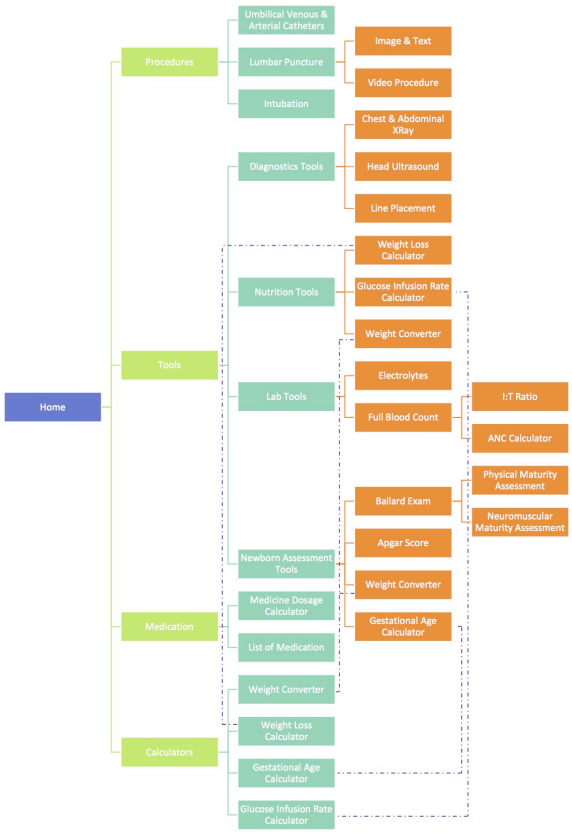


Figure 6: The new Information Architecture.

4. IMPLEMENTATION

There are two general paths to take when creating a mobile app. The first is to create a distinct app for each platform being supporting. This means coding the app in the language and using APIs native to that device. The other path is to code the app in HTML and JavaScript, two languages which most modern browsers -- including mobile browsers -- support. The user then loads the URL of the app in a browser and it will render on multiple devices. Each method has its pros and cons, native apps run faster and smoother but HTML/JavaScript apps take less time to develop. For the sake of time we chose to develop our app in HTML and JavaScript.

Our code base breaks into two parts, which are shown in Figure 7. The first part is an XLSX spreadsheet and the second is our custom HTML and JavaScript code. We will explain each part below but for now notice that both parts are eventually combined into a single ODK app. This ODK app is hosted on Heroku [8] but it could be hosted on any server that can serve HTTP requests.

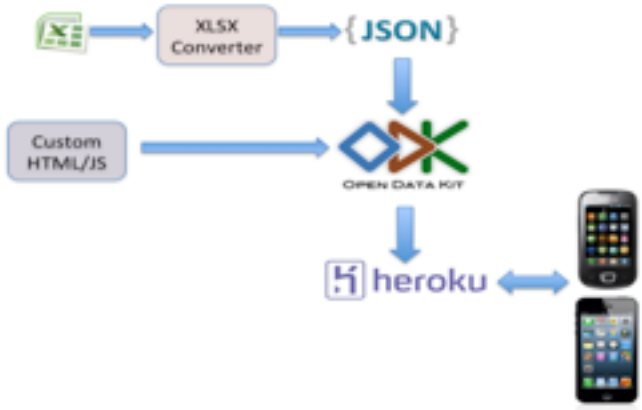


Figure 7: System Architecture

The first section of our code is an XLSX spreadsheet written in the ODK Survey language. ODK Survey defines syntax within a spreadsheet; the XLSX converter (see Appendix 7.1.3), also part of the ODK suite, can compile this spreadsheet. The output of the XLSX converter is a json object, as shown in figure 7. The next step is to set ODK to load the json object and host ODK on a server. For some applications these steps are sufficient, however our app needed more customization.

To realize the requirements that ODK does not support, we implemented some custom HTML and JavaScript, the second part of our code base. The custom code was itself split in two parts: first, custom HTML and Javascript pages; second, modifications to the ODK source code. ODK is incredibly extensible and allows for custom web pages to be inserted and to attach JavaScript controllers to those pages. This means that any functionality that can be implemented in HTML and JavaScript can be inserted arbitrarily into an ODK app. An example from our app is the image slider (more details below), which allows the user to scroll through images and select one, functionality not supported by ODK but easy to add as a custom HTML and JavaScript page. The ability to add custom web pages does not, however, allow for changing the navigation flow of ODK so we also needed to modify the ODK source code. Modifying the ODK source code is to be avoided since it makes upgrading to new versions of ODK more difficult. However, if an app needs different navigation than ODK provides the source code must be edited. Understanding the ODK source code is difficult because it does not have complete documentation. Therefore we needed to consult the lead engineer, Mitch, on many of these changes.

With the json object created by the XLSX converter, our custom pages, and our modifications to ODK we have our complete app. At this point, there are two options for deploying our application. The first is to host it on the web, which allows cell phones to access it through their web browser. The second is to create a native app wrapper around our project -- so that the core code remains the same but the user accesses it by downloading an app from their device's app store. The first option is trivial, the app can be hosted on any server that serves HTTP requests, but the second option is more difficult. Despite the added complexity the payoff of creating a native wrapper is high: it would decrease the latency of pulling pages from our server and it would let users find our app by searching the app store. We did not, in the end, implement our application to run on a native device. This would be a great place for future students to focus their energies.

Gaetano Borriello, founder of ODK and our advisor, has described using ODK for our project as a "stress test" on this project. This is because our project does not conform to the primary use case of ODK: to support mobile surveys. While surveys are the main purview of ODK it has been equipped with navigation controls and database support that make it attractive to any app designer. The idea behind our project was to see if ODK code be tweaked enough to support non-survey apps. In this section we will discuss the process of modifying ODK to suit our needs. As we will see, some things could be achieved by inserting our custom pages into ODK while some things required a change to ODK source code. In the end, we have shown that ODK can be used for a menu-based app.

4.1 Existing ODK Framework

As stated above, most of the functionality of our application comes from the ODK Framework. In this section we will describe how we used and adapted ODK as well as where we needed to work completely outside of ODK. The ODK Framework helped us to manage the overall navigation of our application. Please refer to Appendix 2, which describes the organization of our spreadsheet. Besides providing navigation, ODK provided us with some prompt types such as "notes" and "select_one". The ODK "note" type displays basic html, however, for our application we needed to add a big chunks of information onto single pages. To avoid adding all the text into the XLSX form, we used handlebars templates, which allowed us to add html content and organize it in one file (Appendix 5). That ODK has the ability to add external handlebars is a testament to its' extensibility.

The ODK provided "select_one" type allows the user to select a single choice from a list. We modified this to act as a menu. Please see Appendix 3 for a brief overview of how the menu prompt type extends the existing capabilities of ODK; see the source code for the complete class definition.

Another feature that comes with ODK is the "calculates" sheet on the spreadsheet. This optional sheet allows us to create different types of calculators required for neonatal nurses. To implement the calculator, one uses JavaScript formulas that can be referenced elsewhere in the spreadsheet. Please see Appendices 4 for a more thorough discussion of the calculator feature.

Finally, ODK manages the database throughout the life cycle of our app. Using the database from within a prompt is simple; the prompt must call "this.setValue". The only difficulty is that this is an asynchronous function, which means that success and failure functions must passed as parameters to this function. For an example please refer to Appendix 3, which shows the menu prompt type calling the setValue method of the database. Retrieving data is easier since it is synchronous and takes no parameters, simply call this.getValue() from within a prompt type.

4.2 Linear Image Selector

First we will look at a feature that was implemented with custom HTML and JavaScript, no change to ODK source code was needed. As we stated above this is preferable since it makes upgrading to new versions of ODK easier. The feature we added was a linear image selector (see Appendix 7). ODK already supported image selection, but only through a grid layout. On a mobile device the images, when rendered, were small and there was no way to enlarge them. In our use case a nurse is comparing some visual evidence to the images and needs to be able to see as large a picture as possible. Our solution is to add a new prompt type that displays the images, taking up as much of the device's screen space as possible. ODK allows a developer to add a new *handlebar*, a template HTML page, and attach JavaScript to control the page. We used this feature along with ODK's database layer to complete our linear image selector. In this case we were able to add our feature without changing ODK's source code, however modifying the ODK navigation controller required a change to the ODK source code.

4.3 Navigation Changes

ODK provides a simple and elegant navigation controller. The XLSX spreadsheet defines an ordered list of prompts; prompts may become pages that the user interacts with or they may be control elements like gotos and grouping statements. The application flow begins at the top of the spreadsheet and moves linearly down based on user input, interpreting each new prompt it encounters. The control flow can be interrupted by gotos, which may send the user to arbitrary points in the spreadsheet. Also, multiple prompts may be grouped into a "screen" which aggregates the prompts vertically, the first prompt appearing above the second prompt and so on. There are only two navigation controls available to the user: forward and backward. Backward navigation moves to the previous prompt that the user experienced, not the previous prompt on the spreadsheet (remember the nonlinear application flow). Forward begins a linear search of the spreadsheet, looking for the next applicable prompt. A prompt is applicable if it has no condition or its' condition evaluates to true. This navigation flow works well for surveys, which inherently have a beginning and end and some semblance of order, it did not work well for our app, which is menu based.

To provide a menu based app experience to our users we had to substitute our navigation for ODK's stock navigation. To this end, we created our own prompt type and modified the ODK source code. As an example of adding custom code and modifying the source code, the menu navigation is a good case study of implementing a new feature in ODK. The custom code involved extending an existing prompt type in ODK (the *select_one* prompt) and changing some of the behavior, e.g. removing the stock ODK *next* button and rendering the next screen when the menu item is selected. The fact that we could easily add a new feature by extending an existing prompt reflects ODK's extensibility. At this point, however, we encountered a new problem. Once the user arrived at a non-menu page they could still press the *next* button taking them out of our desired navigation and back to ODK's stock navigation. Thus we had to modify the ODK source code to remove, completely, the *next* button, giving us full control of the user's experience navigating our app. In light of our implementation of menu navigation, ODK appears highly extensible and applicable to problems outside of traditional survey. More extreme extensions to ODK should be done to see just how far it could be pushed.

5. EVALUATION

We have three levels of evaluation for our app. The first level is the Human Centered Design Engineering (HCDE) design team. They have designed the app so they have a good idea of what to

expect. For this reason, they cannot give unbiased feedback on the overall design of the app but they can give feedback on small design decisions we made. For instance, we might decide to veer from their design because some other implementation was far easier or they may have left out some details of the design, which we decided to fill in on our own. For example, the original design of the Ballard Exam allowed two nurses to complete the exam and compare results. We limited the tool to one nurse since it made implementation far easier. In both cases we would run the decision by HCDE first, before showing the app to outside testers.

The next level of evaluation is the originator of the idea, Jocelyn Kirk. Jocelyn can provide feedback almost as good as the target client since she is herself a neonatal nurse. She knows what nurses need and can role play the part of a nurse during testing. However, since her previous experience with the app may color her feedback, she cannot be relied upon as much as a neonatal nurse who has never seen the app. Despite this downside she is a valuable resource as she is a neonatal nurse who is readily available to us.

The third and most important level of feedback comes from neonatal nurses, they are the intended client of the app. Finding nurses to test our app can be difficult. Since we cannot impose on their private lives, we have to schedule tests while the nurses are at work, which means getting permission to conduct the test in a hospital. We have managed to do this once and discuss the results below.

Midway through implementation of our application, we travelled with our HCDE design team to the University of Washington Medical Center (UWMC). Here, we conducted our first usability test with nine nurses – eight NICU Registered Nurses and one dietician. We asked the nurses to evaluate both our high fidelity mock up as well as our partially implemented application. The results of the usability test caused the HCDE team to redesign the layout of the app. We will note two examples of design features that changed based on the feedback from the usability tests. The first is the main menu, which originally had 8 categories. The nurses commented that it was too cluttered and confusing so we reduced the number to four categories. Also, the nurses did not like the text menu buttons so the HCDE designed graphical elements to serve as the menu items. The pretest and posttest menus are shown below (see Figure 8a). The second change made with the feedback from nurses was the direction in which to present images. We originally presented images horizontally, allowing the user to swipe left or right to see new images. The nurses preferred all images on a single page with vertical swiping (see Figure 8b).

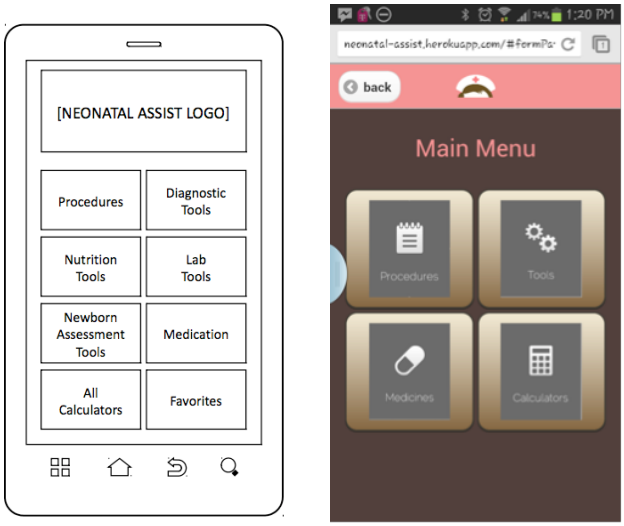


Figure 8a: Pretest and Posttest Design.

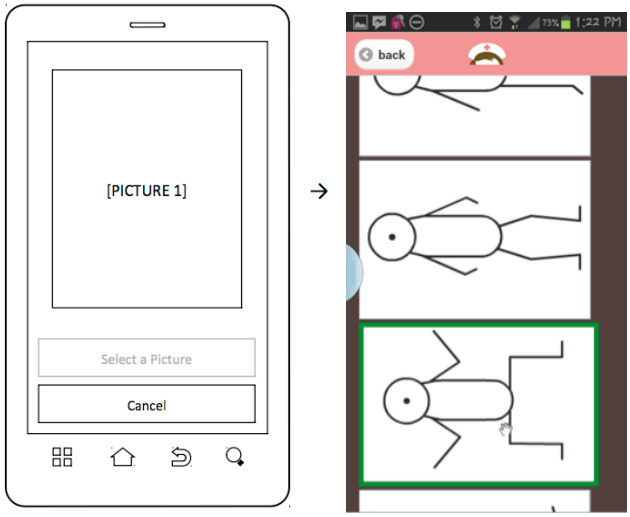


Figure 8b: Pretest and Posttest Design.

Since we are unable to conduct another usability test in the time allotted, we will provide qualitative and quantitative metrics, which can be used by future students who continue this project. The participants will need some direction and so we decided to create three tasks for them to complete: administer the Ballard Exam, calculate the gestational age of an infant, and find resources for chest x-rays. To gather quantitative data, we will record three metrics for each task: time to complete task, unnecessary clicks, and the number of times we had to help them through the task. The goal is to design an app, which minimizes all three of these metrics. As for qualitative data, we are using some of the questions generated by the HCDE team.

1. Was there something missing that you were expecting to see? If so, can you tell us what was missing and, if possible, why it should be there?
2. Were any features or labels confusing to you? If so, which ones were confusing?
3. Did you understand how to navigate through the app?
4. How intuitive is the navigation system from 1 to 10?

5. Would you use an app like this if one were available? Why? If you would use the app, would you use it at work?
6. What did you like most about the app? Why?
7. What did you like least about the app? Why?
8. If you could change one thing on the app, whether it is major or minor, what would be at the top of the to do list?
9. Do you have any feedback or suggestions you would like to give us regarding the app?

6. CONCLUSION AND FUTURE WORK

As a conclusion, we will examine how far we have come in completing our goals, what is left to be done, and what we learned along the way. Our main goal was to provide a resource to neonatal nurses that would have the right blend of content in an app that would be easy to navigate. To provide an example of the app's potential we implemented representative components. For instance we provided calculators for a few of the medications that the nurses need, an exam, and some reference material including text and video. These examples together do not constitute an app that nurses would want to use since they are not a complete reference. However, they can provide enough information for us to make changes along the way to finishing the app. With that in mind, our team conducted in person usability tests with neonatal nurses at the UWMC. We used their feedback to redesign our app; the redesign has not been tested and this would be an important next step.

One sub goal of our project was to test the ODK framework and provide feedback on ODK. The test we were conducting was to see if ODK could be extended to fit a non-survey app and what roadblocks we would run into. One issue was bypassing the first couple screens of ODK, which set up an instance of a survey. This was not simple to do but we were able to achieve this goal by changing some of the builder code for ODK. In the end, we believe that ODK can be extended to non-survey apps, however if this is a goal of ODK, it should provide more customizability in navigation control.

There exists plenty of work to be done for our app. First of all, there is a lot more content to add. We could start by extending the current list of medications we support to include all of the most common medications used by neonatal nurses. We could also increase the number of procedures and exams included in our application. This task is easy since we have already implemented examples of each category, however, it is time consuming. This raises another question: who will implement these additions to the app? We hope that we have provided sufficient information in the Appendices to support future work.

There are a few more technical challenges we leave to future groups. One challenge is how to use the existing Android Survey app to wrap our project in Android. A second challenge is to upgrade to the new version of ODK, which should be educational since ODK uses an asynchronous callback structure very common in real web apps. Finally, an interesting challenge would be to compensate for the different protocols at different hospitals.

7. APPENDICES

7.1 Reconstruct Neonatal Assist

7.1.1 Host Locally

- Clone this project from https://github.com/xiaoj/481web.git. Code is included in this package as well.
- Open a terminal and cd to the project root.
- Run the following command: ./node app.js (do not close the terminal window).
- Open browser and navigate to localhost:8888. Note: you must clear your browser's cache and saved app. data every time the app changes.

7.1.2 Files to Note

- /app.js - This is a node.js app which redirects a user to the odk app.
- /default - This directory contains the code for ODK. You can learn more about ODK: http://opendatakit.org/use/collect/
- /neonatal - This directory contains the code specific to our app. You can learn more about the architecture of our app from our paper.

7.1.3 Convert Spreadsheet to JSON

- cd into project root.
- Run ./phantomjs convertPhantom neonatal/neonatal.xlsx > neonatal/formDef.json

7.2 Layout of XLSX Spreadsheet

The layout of Neonatal Assist Application is in the Excel Form named "neonatal.xlsx". This spreadsheet defines the content and navigation flow of our application. There are currently 5 sheets in the spreadsheet including survey, choices, calculates, prompt_types, and settings.

- The survey sheet is the main layout of the application.
- The choices sheet contains different lists of items specified by unique name.
- The calculates sheet contains calculation formulae.
- The prompt_types sheet is for specify additional prompt types.
- The settings sheet contains form id, form title, etc.

For more information of ODK Survey:

http://code.google.com/p/opendatakit/wiki/XLSForm2Docs#survey

7.2.1 Example Survey sheet

line	type	name	label	condition
1	menu menu	menu	Main Menu	
2	goto procedures_begin			selected(data('menu'),'procedures')
3	goto procedures_end			not(selected(data('menu'), 'procedures'))
4	label procedures_begin			
5	menu procedures_menu	procedures_menu	Procedures	
6	begin screen			selected(data('procedures_menu'), 'intubation')
7	note		Intubation	
8	end screen			
9	begin screen			selected(data('procedures_menu'), 'lumbar')
10	note		Lumbar	
11	end screen			
12	label procedures_end			

Below is a description of the procedures menu and sub items under procedures:

Line1 “menu menu”: First “menu” represents the menu prompt type. Second “menu” represents the list of item of “menu” under choices sheet (section 1.2).

Line 2 “goto procedures_begin”: If the condition in line 2 is true, then the application will go to line 4 label procedures_begin. Otherwise, the form proceeds to next line.

Line 3 “goto procedures_end”: If the condition in line 3 is true, then the application will go to line 12 label procedures_end. Otherwise, the form proceeds to next line.

Line 4 “label procedures_begin”: This is a unique label, which delineates our procedures category. This helps us to organize the spreadsheet.

Line 5 “menu procedures_menu”: “menu” represents the menu prompt type. “procedures_menu” is a unique name that contains a list of item of the procedures menu under choices sheet (section 1.2).

Line 6,8 “begin screen”, “end screen”. This is a prompt type used to specify that anything between begin screen and end screen are grouped onto a single screen when rendered.

Line 7 “note”: Display html content.

7.2.2 Example Choices sheet

list_name	*name*	*label*
procedures_menu	intubation	Intubation
procedures_menu	lumbar	Lumbar

- list_name: The list_name is used to group choices into lists that can be referenced from the survey sheet. The example above could be referenced as follows
- name: The name sets the value that the choice is stored as. This is useful in constraint formulas, and for formatting exports.
- label: What the user will see.

7.2.3 Example Calculates sheet

The calculates sheet is an optional sheet that allows you to create JavaScript formulas that can be referenced in other formulas. It has two columns: name and calculation. The name column sets the name used to reference the calculation (e.g. calculates.name()), and the calculation column defines the JavaScript formula to be evaluated.

If you need to a more complex calculation that defines temporary functions/variables you can wrap it in an immediately invoked function like so:

```
(function(){
  var add = function(a, b){
    return a + b;
  };
  return add(1,1);
}())
```

7.2.4 Example Prompt_types sheet

name	Schema_type
Image_slider	String
Menu	String
ballard	String

- name: Name of the prompt type.
- schema_type: JavaScript type of the prompt type.

7.2.5 Example Settings sheet

setting	value
form_title	Example Form
form_id	example

The settings sheet has two columns: setting and value. Additional columns for Internationalized values (e.g. value.hindi) are allowed for some settings, such as the form title. These are the available settings:

form_title	The name of the form that is displayed to users.
form_id	A unique identifier for the form.
table_id	The id of the table that form data gets stored in.
font-size	A css font-size that sets the form's global font-size.
theme	The name of a pre-packaged Survey theme to use. (Currently can be default or square.)

7.3 Create Menu Based ODK App

Below is an example of how we modified ODK to suit the navigation requirements of our app. This following code is a custom prompt type called “menu” and can be found in “neonatal/customPromptTypes.js”:

```
modification: function(evt) {
….
 this.setValue($.extend({}, ctxt, {
        success: function() {
          that.updateRenderValue(formValue);
          that.render();
          controller.gotoNextScreen(ctxt);
        }
 }), this.generateSaveValue(formValue));
…
}
```

7.4 Add new calculator

The source code for our calculators is in neonatal.xlsx, under the section “label calculator_begin”. There are two sheets in excel that users have to change if they want to implement a new type of calculator, the “survey” and “calculates” sheets.

survey sheet:

- Type: the type that user want i.e. decimal is a input box for number.
- Name: distinct name representing the object in that row.
- Label: text you want to display for the object.
- Condition: the object will display if it satisfied the condition

calculates sheet:

- Name: distinct name representing the calculation.
- Calculation: place to put equation for calculation.

Example:

To Implement a Weight Converter Calculator from Pounds to Kilograms:

survey sheet:

Type	Name	Label	Condition
begin screen			

decimal	weightInPound	Enter weight in pounds:	
note		Weight is: {{calculates.convertPoundToKg}} kg	data('weightInPound')
end screen			

calculates sheet

Name	Calculation
convertPoundToKg	data('weightInPound')/2.2

7.5 Add new page

To add new content into a page

- Create a file called <your_note_name>.handlebar and put it in the templates folder.
- In the <your_note_name>.handlebar, below is the template:

```
<div class="{{#if hide}}odk-hidden{{/if}}">
  {{> labelHint}}
  Add any content you want using html based format.
  ...
</div>
```

- In neonatal.xlsx, under the section where you want the content to appear.

type	name	label	condition	templatePath
note				templates/<your_note_name>.handlebar

- Reconvert formdef.json
- Delete your browser's cookies then open the Neonatal Assist Application.

7.6 Debug ODK with Chrome

- Open up chrome dev tools and put a breakpoint in the render method of the prompt type you are working on. Other possible breakpoints include anywhere in the prompt type you are working on as well as the gotoNextScreen method in screen controller. This last point lets you see what the state is in between the two prompts to try and identify where the problem is.
- Look at the $el field in the variable "this" and watch how it changes as you go through your code.
- For syntax errors click the little red x in the bottom right of dev tools. There should be about 3 errors that just have to do with not finding style sheets or icons etc. Ignore these. See if there are any other errors.

7.7 Vertical Image Selector

ODK does currently support image selection through a select type with image choices. This will create a grid or list of images, which the user can select. The benefit of this prompt type is that it is already implemented and not any harder to use than a normal select. However, a major downside is the size of the images that are displayed -- they are thumbnails and there is no way to enlarge them. In our use case, and probably others, the user is comparing some visual evidence to the images and needs to be able to see as large a picture as possible. Our solution is the image_slider prompt type. It acts like a select_one except that the images fill the entire screen. Touch slides are used to

navigate through the photos and a single touch event will select a photo. The 'name' of the photo is stored in the database, just like a select_one type.

There are some potential pitfalls to the specification of the image slider. As we said above, the slider takes the user "back" to the previous screen. This works well in our app since we are implementing a menu structure. So in our use case a user selects from 6 items, each of which has an image slider; when they are done with one they go back and choose another, navigating in arbitrary order. If someone wanted to use the image slider as part of a traditional linear ODK Survey, it would be difficult if not impossible. A reasonable solution would be to implement a setting that would allow the form creator to choose whether the confirm button brings the user "back" or "forward". Either way traditional ODK navigation would still be disabled i.e. sliding motions affect the image slider not the app navigation.

A downside to disabling ODK navigation is that the user might have grown accustomed to navigating using the slide motion. When they come to the image slider this motion will act differently than the rest of the app, possibly causing annoyance or confusion: all of the sudden the user will have to use buttons to navigate.

7.8 Information for Making Tab Based ODK Page

- Create a new prompt type called "screen_tab", model it after the screen prompt type in prompts.js:1644
- In the xlsx spreadsheet use "begin screen_tab" and "end screen_tab" like with "begin screen" and "end screen"
- What you put in between "begin screen_tab" and "end screen_tab" should be available in the prompt type as this.prompts.
    - You can put a series of notes in the screen_tab and just put whatever you need in those notes.
    - You might even be able to put a "begin screen" in a "begin screen_tab" but I am not sure.
- To build up the page append stuff to this.$el in the prompt type. this.$el is what gets emitted by odk.
- You will implement the tab feature in the "render" method of the prompt type.
    - Use jquery UI tabs which you can learn about here http://jqueryui.com/tabs/
    - You can loop through this.prompts and call render on each prompt. Then take what the prompt renders and append it to this.$el
        - To get the html from the prompt, call render on it then access its $el field.
- You can basically copy the screen prompt in prompts.js:1644
    - Just change the render method to emit jQuery tabs instead of what it does currently.

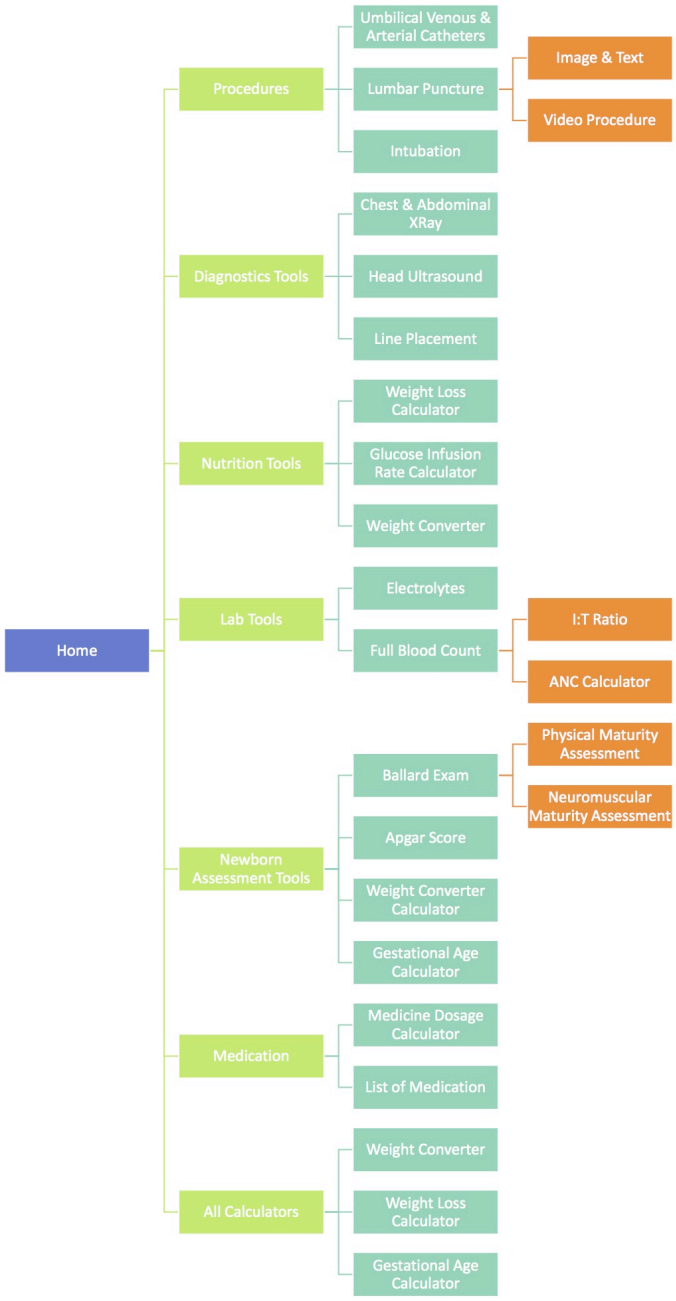
7.9 Change to ODK Source Code

- In prompts.js moved afterRender call to the imagesLoaded function, which is in the render function of promptType.base. This was a bug in ODK that didn't allow us to call unique javascript code in our afterRender function of our custom prompt type. Conferred with Mitch and this was his fix, he fixed it in master ODK branch as well.
- We wanted a menu structure that forced people to click items instead of "next". To achieve this we set "enableForwardNavigation" to false in screen manager
- We bypassed the opening two screens for ODK survery, so that the user comes straight to our main menu. This consisted of changes in to places in the source code. One is in Builder.js line 237 where the opening prompts are defined. We send the user to a custom prompt "silent_create". The source code for "silent_create" is in our modified prompts.js.
- The swipe was too sensitive so we added
  $.event.special.swipe.horizontalDistanceThreshold = 120; // (default is 30)

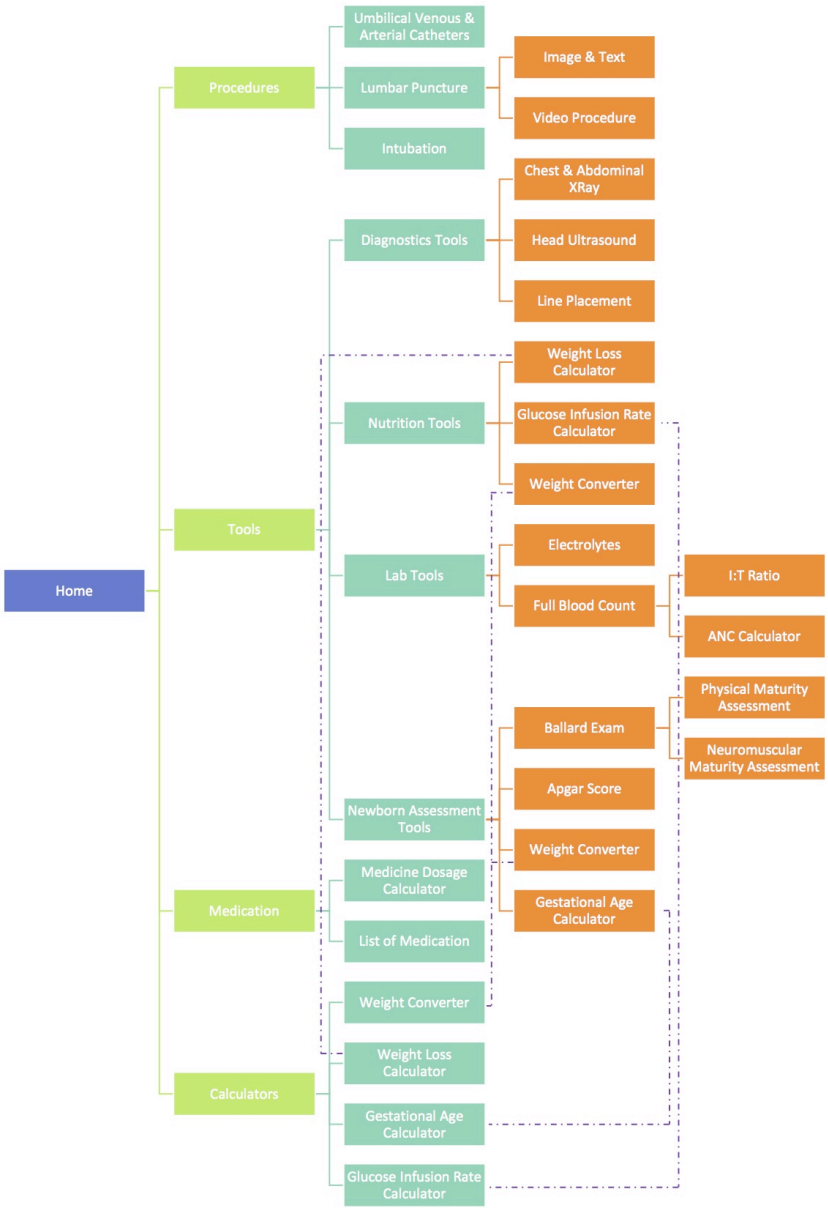
7.10 TABLE 1: List of Related Work

Feature	Neonatal Assist	Pedistat [5]	Neonatal Calculator [6]	SimNICU [7]
Umbilical Venous & Arterial Catheters				
Lumbar Puncture				
Intubation Overview				
Intubation tube sizes, depth etc.				
Chest & Abdominal XRay				
Head Ultrasound				
Line Placement				
Weight Loss Calculator				
Glucose Infusion Rate Calculator				
Weight Converter				
Electrolytes Lab				
Full Blood Count Lab				
Ballard Exam				
Apgar Score				
Weight Converter Calculator				
Gestational Age Calculator				
Medicine Dosage Calculator				
List of Medication				
Cardiac Resusitation				
Anaphalaxis				
Anion Gap				
Serum Osmolality				
Fractional Excretion of Sodium				
Oxegenation index calculator				
Hernia Management				

7.11 History of Information Architecture



Old Information Architecture



New Information Architecture