# APPENDICES

# APPENDICES INDEX

1. Reconstruct Neonatal Assist
2. Layout of XLSX Spreadsheet
3. Create Menu Based ODK App
4. Add New Calculator
5. Add New Page
6. Debug ODK with Chrome
7. Vertical Image Selector
8. Information for Making Tab Based ODK Page
9. Changes to ODK Source Code
10. Table 1: List of related work
11. History of Information Architecture

# Reconstruct Neonatal Assist

## Host Locally

1. Clone this project from <https://github.com/xiaoj/481web.git>. Code is included in this package as well.
2. Open a terminal and cd to the project root.
3. Run the following command: ./node app.js. Note: The terminal will not output anything; it will just display a flashing cursor.
4. Open browser and navigate to localhost:8888. Note: you must clear your browser's cache and saved app.
5. Data everytime the app changes.

## Host on Heroku

1. Clone this project.
2. Open a terminal and cd to the project root.
3. Run the following command: git push.

## 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.
* /Presentation - Contains the final version and supporting files for our presentation.
* /paper - Contains the final version and supporting files for our paper.
* convertPhantom - command line conversion from xlsx to json.

## Convert Spreadsheet to JSON

* cd into project root.
* Run ./phantomjs convertPhantom neonatal/neonatal.xlsx > neonatal/formDef.js

# Layout of XLSX Spreadsheet

**Neonatal.xlsx Overall**

The layout of Neonatal Assist Application is in the Excel Form named “neonatal.xlsx”. There are currently 5 tabs in the excel sheet including “survey”, “choices”, “calculates”, “prompt\_types”, “settings”.

2.1 The “survey” sheet is the main layout of the application.

2.2 The “choices” sheet contains different lists of items specified by unique name.

2.3 The “calculates” sheet contains calculation formula.

2.4 The “prompt\_types” sheet is for specify additional prompt types.

2.5 The “settings” sheet contains form id, form title, etc.

For more information of ODK Survey:

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

**2.1 “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 the example of Procedures category.

Line1 menu menu: First menu word represents the menu prompt type. Second menu word 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 of Procedures Category.

Line 5: menu procedures\_menu: The menu word represents the menu prompt type. The procedures\_menu word 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 specify that anything between begin screen and end screen are group onto a single screen when rendered.

Line 7: note. Display a label with no input.

**2.2: “Choices” sheet**

Below is the example of Procedures choices

|  |  |  |
| --- | --- | --- |
| list\_name | name | Label |
| procedures\_menu | intubation | Intubation |
| procedures\_menu | lumbar | Lumbar |

**Choices sheet columns:**

* **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:

|  |  |  |
| --- | --- | --- |
| **Type** | **name** | **label** |
| select\_one cities | cities\_visited | Which cities have you visited? |

* **Name**: The name sets the value that the choice is stored as. This is useful in constraint formulas, and for formatting exports.
* **label/image/audio/video**
* **appearance**: Sets appearance flags for certain types of prompts.

|  |  |
| --- | --- |
| grid | Show the choices in a grid. |
| inline | Show the question and choices in a single line. |

**2.3: “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 calculate (e.g. calculates.name()), and the calculation column defines the JavaScript formula to be evaluated.

If you need to a more complex calculate 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);

}())

**2.4: “prompt\_types” sheet**

Example of prompt\_types sheet:

|  |  |
| --- | --- |
| name | Schema\_type |
| Image\_slider | String |
| Menu | String |
| ballard | String |

Prompt\_types sheet columns:

* Name: Name of the prompt type.
* Schema\_type: Type of the prompt type.

**2.5: “settings” sheet**

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.) |

# Create Menu Based ODK App

ODK is designed to create linear tree-like surveys. It has great control over the order that prompts appear and in fact it allows you to do loops and re-ask the same question however the “next” and “back” navigation act like your browser and not like a menu. Thus we needed to design a widget that would act like a menu. The menu widget we designed is basically a select\_one that auto-advances and has the next button disabled. The auto-advancing is to remove a step (needing to click “next”) and the disabling of the next button is to keep a menu like hierarchy and keep ODK from imposing its own hierarchy. One problem that arose is interaction between how the select\_one saves the selection and the auto-advancing feature we added. When a user returns to a menu that’s already been navigated, the previous option is still selected. When they click that option again it deselects, instead of the intuitive behavior which would be to visit that section again. To fix this we modified our “menu” prompt type to erase any previous choices when it is rendered. The key change to create the menu prompt type is shown below. This chunk of code is executed when the user clicks one of them items in the menu. The success function is a callback which executes after the database has been updated. As can be seen, the call back function calls gotoNextScreen, which is the key line in auto advancing. Of course, the entire menu prompt type can be viewed in the source code.

modification: function(evt) {  
….

this.setValue($.extend({}, ctxt, {

success: function() {

that.updateRenderValue(formValue);

that.render();

controller.gotoNextScreen(ctxt);

}

}), this.generateSaveValue(formValue));

…

# Add new calculator

The source code of our calculator is in neonatal.xlsx, under the section “label calculator\_begin”. There are two sheets in excel that users has 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 |

# Add new page

**5.1 To add new content into a page**

5.1.1. Create a file called *yournotename*.handlebar and put them in templates folder.

5.1.2. In the *yournotename*.handlebar, below is the template:

<div class="{{#if hide}}odk-hidden{{/if}}">

   {{> labelHint}}

*Add any content you want using html based format.*

*…*

</div>

5.1.3. In neonatal.xlsx, under the section you want the content in *yournotename*.handlebar to appear.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| type | name | label | condition | templatePath |
| note |  |  |  | templates/*yournotename*.handlebar |

5.1.4. Regenerate formdef.json

5.1.5. Clean the cookie of your browser, then open the Neonatal Assist Application.

# 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.

# 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, which are displayed -- they are thumbnails and there is no way to enlarge them. In our use case, and I assume 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 double click or touch-hold event will select a photo. The ‘name’ of the photo is stored in the database, just like a select\_one type. One technical challenge was that the slide motion would be handled by ODK survey (in this context it represents a navigation motion through the survey i.e “go to next prompt”). Our solution was to disable navigation for this prompt type, thus allowing the image\_slider to catch the slide events and scroll through the images. To replace the navigation we placed a “confirm” and “cancel” button, both of which take the user back to the previous screen.

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.

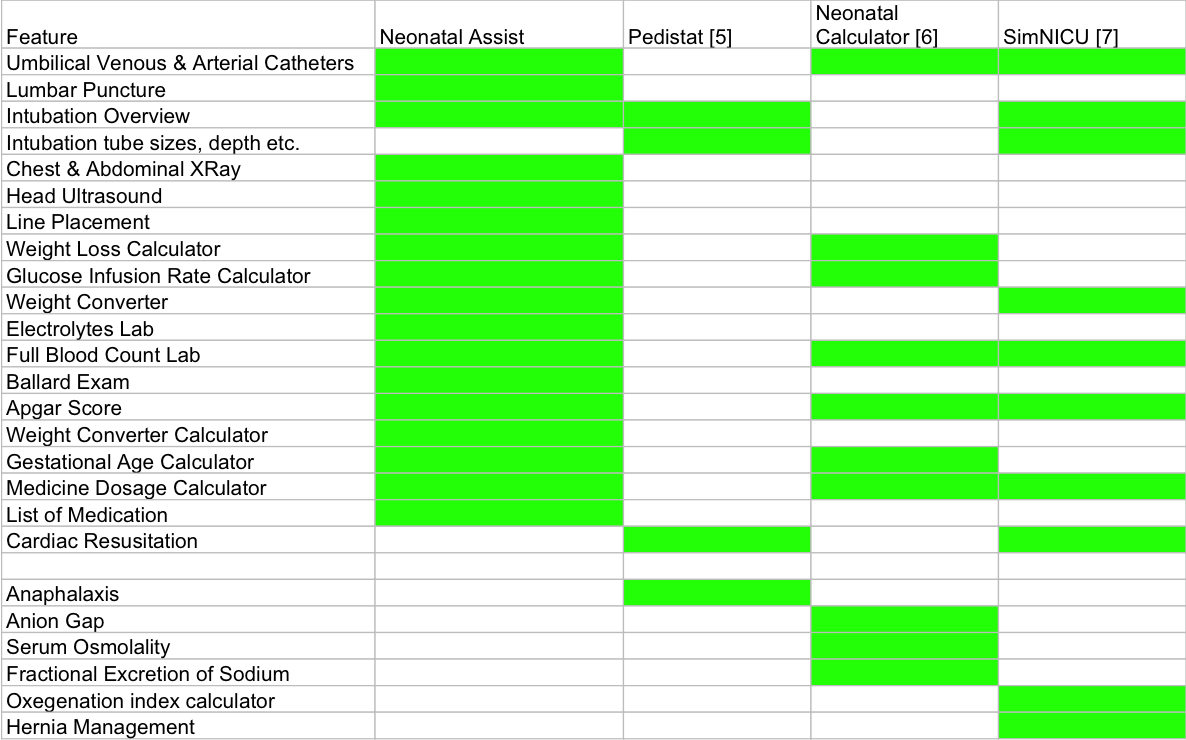
# Information for Making Tab Based ODK Page

* 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.

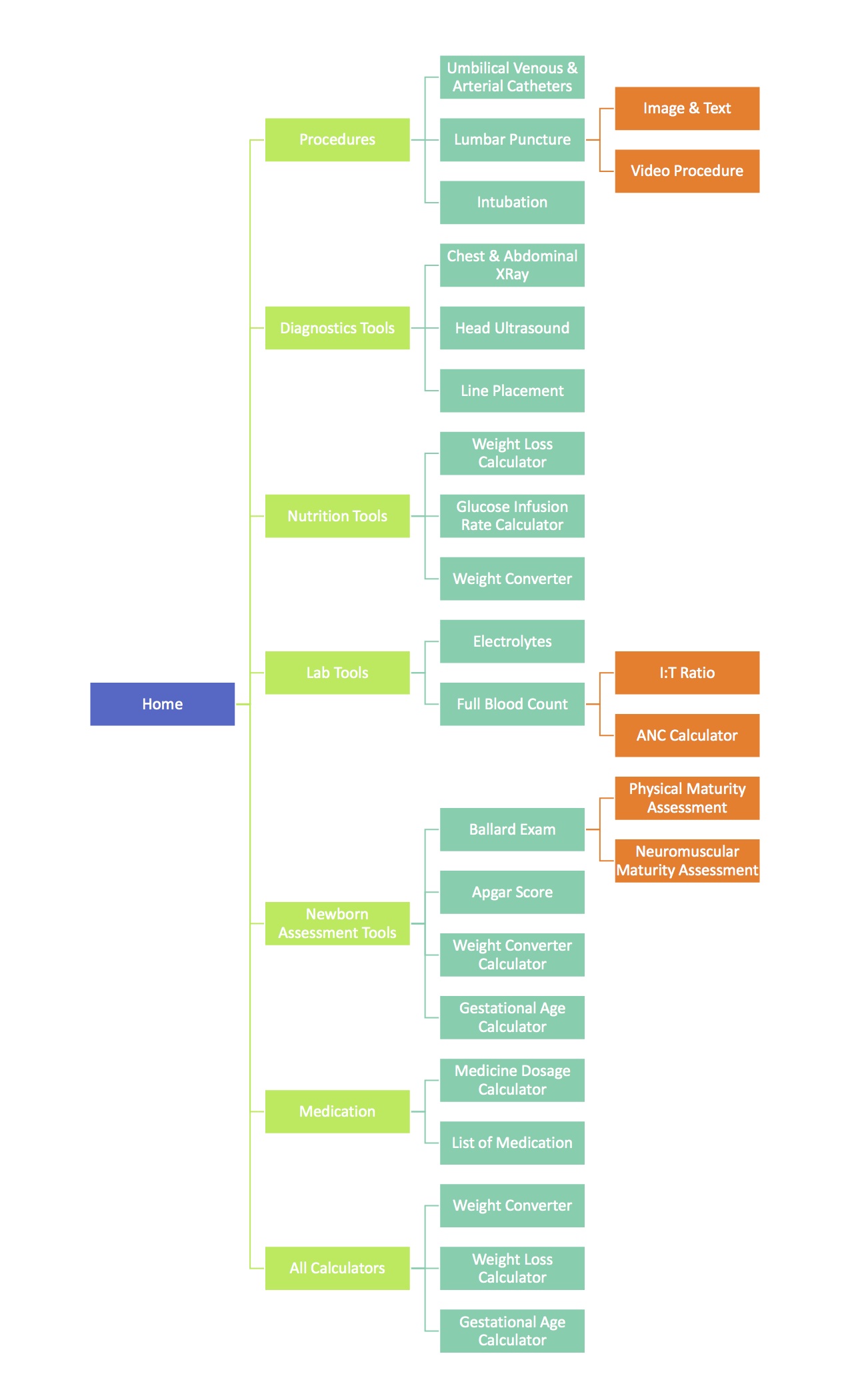
# Change to ODK Source Code

* 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. This had the unintended consequence of making the “opening” screen loose it’s next button, thus you couldn’t start the survey. To fix that we set “enableForwardNavigation” to true in the opening prompt type. We will talk to Mitch and see what he thinks about this.
* The swipe was too sensitive so we added   
  $.event.special.swipe.horizontalDistanceThreshold = 120; // (default is 30)

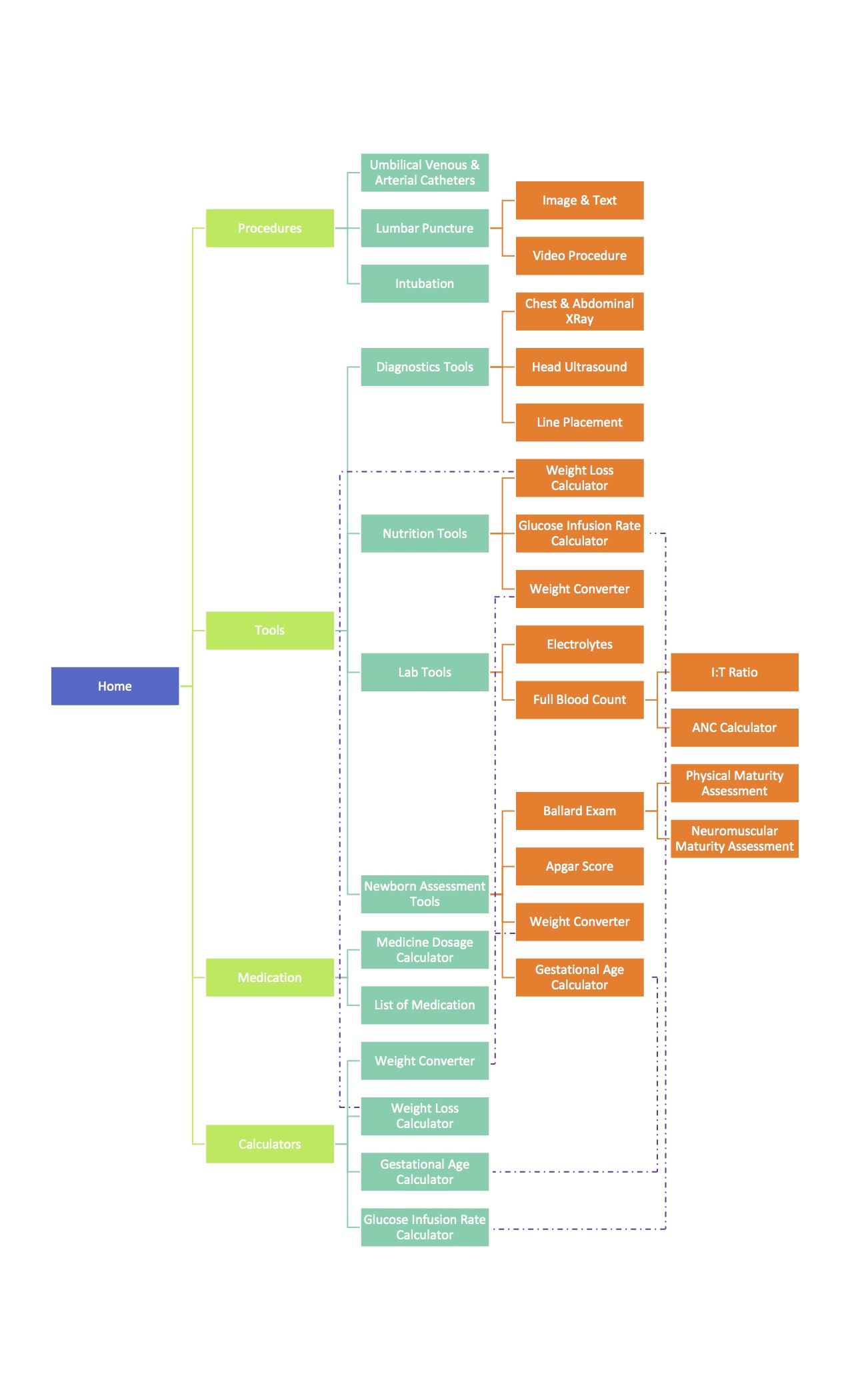
# TABLE 1: List of Related Work



# History of Information Architecture



Old Information Architecture



New Information Architecture