SIMPLE APP-DEVELOPMENT PROCESS

1. PROBLEM ANALYSIS: (PAPER AND PENCIL PHASE)

What am I trying to accomplish? If I think about it as system, what are the entities and process involved? What are the rules that govern the system? Do I have all the information I need?

EVALUATE THE PROBLEM: What am I being asked to do? Are the instructions clear, or is there ambiguity? What are the constraints, and are they clear? What are the inputs, and how are they captured? What are the outputs, and how are they delivered? Are there data storage requirements? If so, how is data to be stored -- on the client side, or on the server?

If the problem is stated in a long block of text, try breaking it apart into a list simple statements or instructions. It there is ambiguity, try rewriting the problem statement more clearly.

DEFINE THE ENTITIES: What are the elements or entities of the problem? Are they well defined? If not, what do I need to know in order to solve the problem?

Use lists and simple diagrams to sketch out the entities that make up the system. Detail the key attributes of each. If key information is missing, make special note of what needs to be known.

DEFINE THE PROCESSES: What discrete processes are involved? Are they well defined, or is there ambiguity? How many processes are there? Are they independent, or do they rely on one another? If processes are dependent, what are those dependencies? In what order must things occur?

DEFINE THE RULES: Are there rules that govern the behavior of the various entities and processes?

Key Techniques:

Paraphrase and refine problem statements.
Make a list of specific inputs and outputs.
Make a list of entities and their key properties.
List discrete system processes and indicate dependencies with other process.
Write out all requirements and business rules.

2. SOLUTION MODELING (PENCIL AND PAPER OR MODELING SOFTWARE)

What is the structure of my solution, and how does it behave? How do I best represent my solution with symbols?

Software modeling systems can be quite involved. UML (Unified Modeling Language), for example, has more than a dozen different diagrams types, each serving a specific need. For our purposes though we will use just a few diagrams in a two-part approach to creating a model.

MODEL THE STRUCTURE OF YOUR SOLUTION: The first part of the model is focused on Structure. You want to define all of the objects you're going to use in detail. You do this by translating the Entities defined in the Problem Analysis phase into formal Object Diagrams (see diagram examples for details). Every object has three parts: name, attribute properties and method properties.

MODEL THE BEHAVIOR OF YOUR SOLUTION: The second part of the model is focused on the behavior of the system. The behavior model will translate entities, processes and rules of your solution into formal models. In this example you will use two diagram types: the Sequence Diagram to provide a broad global view of the system behavior; and the Flowchart to provide a more granular view of individual processes (see diagram examples for details).

3. CODE BLOCKING & PSUEDO-PSEUDO CODING

How do I translate my solution model into a well structured program?

Divide and Conquer is the key to solving big problems. Code Blocking and Pseudo-Pseudo Coding lets you break big problems into a series of smaller, bite-size problems. It works by combining code structures and comment blocks to form a skeleton-like framework that serves as the foundation of your program. It may help to think of it like the framing of a house.

For the record, the system I describe here differs from what is commonly referred to as pseudo-coding, thus the name, *pseudo-pseudo* coding. In conventional pseudo coding, you type in all of your program statements, but instead of using actual programming language syntax, you create statements that use something close to everyday language. This, for example, is what Fortran pseudocode might look like:

```
program fizzbuzz
Do i = 1 to 100
    set print_number to true
    If i is divisible by 3
        print "Fizz"
        set print_number to false
    If i is divisible by 5
        print "Buzz"
        set print_number to false
    If print_number, print i
    print a newline
end do
```

There may be a good use for pseudo-coding, but frankly it adds another step which may not be necessary for anyone familiar with programming. Pseudo-pseudo coding, on the other hand is really about defining the larger structure that will be your program. It speeds up your coding rather than adding an extra step. Pseudo-pseudo coding…

1. lets you easily visualize how the larger solution model can be broken up into a series of smaller problems (objects, functions and variables).
2. helps you prioritize your activities and makes it easy to focus on smaller tasks.
3. makes refactoring easier if there are significant design changes.

So how do you generate pseudo-pseudo code? While there is no one way to create pseudo code, here are some general guidelines.

1. Open up a file in you IDE or script editor.
2. Define the modular level variables that you think you will need, and comment each one.
3. Define the object variables that you think you will need (refer to your Object Diagrams), and comment each one.
4. Declare as many functions and you can think of without worrying about the function parameters or any of the function statements. Include a comment block above each function that states its purpose.
5. You are going to have an interface that will drive event handlers in the code, so create a special initialization function shell. Wire it to the window onload event or whatever event you want to trigger the initialization process.
6. With this complete you have a basic structure for program. Now move to your initialization function. Write a commented line for each step of your initialization

process – just a simple descriptive statement of a single simple action that the program is to take.

7. With the initialization function complete, go to the objects in your code. Key in the attribute properties (object variables) for each object and include a comment statement for each. Then create the shells for the function properties (object methods), and include a comment for each describing it's purpose.
8. With the skeletons of your objects in place, move through the functions of your solution one at a time. Code in the local variables that each function will use, and comment each. Then create a series of descriptive comments within each function that walks through the processes that make up the function. As you are working through this process you may begin to define parameters and calls to other functions or object methods that return values.

Here is an example of pseudo-pseudo code for a simple app used to display images in an overlay on the page. This particular project required that I use an IIFE (Immediately Invoked Function Expression) to return a single literal object that would contain a series of methods that would display overlaid images when a link on the page was clicked.

First Snapshot:

```
var Preview = (function(){
  var Preview = {};

  Preview.init = function(){
    console.log("init");
    //get all the a tags and collect all those with the attribute "data-preview"

      //loop through the collected links and assign an onclick function handler that

        //cancels the default action

        //gets the url for the image

        //creates a pop up div that

          //disables the controls on the page

          //creates and displays an image previewer element

          //loads the given image into the preview

             //and creates a close button which is assign its own event handler that

               //unloads the images by hiding the div and enabling the controls on the page

    };
    return Preview;
})();
```

Once the basic structure of your pseudo-pseudo code is in place, you can begin working back and forth between pseudo-coding and writing actual code. As the program is refactored, you can mode blocks of pseudo code around, as see here:

Second Snapshot:

```
//public gloabal variable, Preview, holds the module to manage the image preview
var Preview = (function(){

    //literal object that's returned and assigned to the global variable of the same name
    var Preview = {};

    //the one public method for this Preview API
    Preview.init = function(){

            //get all the a tags and collect all those with the attribute "data-preview"
            var links = getImageLinks();

            //loop through the collected links and assign an onclick function handler that
            for (var i = 0; i < links.length; i++){

                    links[i].onclick = function(){
                    //cancels the default action

                    //gets the url for the image


                    }
            }

    };

    //function retrieves all of the page links that have images to preview.
    function getImageLinks(){
            var links = [];
            return links;
    }

    function showPreview(url){

      //creates a pop up div that

      //disables the controls on the page

      //creates and displays an image previewer element

         //loads the given image into the preview

             //and creates a close button which is assign its own event handler that

               //unloads the images by hiding the div and enabling the controls on the page

    }

    return Preview;
})();
```

As you can see the program begins to take shape around the pseudo code. The pseudo code eventually becomes the comments for the program that takes shape around them. In this example, a couple of new functions have appeared, one that I know how to solve pretty easily (getImageLinks) and one that I'm not yet sure how I'm going to handle just yet (showPreview) as seen in the third snapshot.

Third Snapshot:

```
//public gloabal variable, Preview, holds the module to manage the image preview
var Preview = (function(){

    //literal object that's returned and assigned to the global variable of the same name
    var Preview = {};

    //an array to hold the image links, indexed by the data-preview attribute
    var images = [];

    //the one public method for this Preview API
    Preview.init = function(){

            //get all the a tags and collect all those with the attribute "data-preview"
            var links = getImageLinks();

            //loop through the collected links and assign an onclick function handler that
            for (var i = 0; i < links.length; i++){

                    //add the image url to the images array

                    //assign an event handler to the a-tag element's onlick event
                    links[i].onclick = function(e){

                            //cancels the default action
                            if(e.preventDefault){
                                    e.preventDefault();
                            }

                            //gets the url for the image
                            showPreview(links[i].getAttribute("href"));

                            //return false to cancel the default event (for browsers that don't
implement the prevent default method)
                            return false;
                    };
            }

    };

    //function retrieves all of the page links that have images to preview.
    function getImageLinks(){

            var els = document.getElementsByTagName("a"), //a collection of all the a a-tag
elements on the page
                    links = []; //an array to hold all the a-tag elements that link to images

            //loop through the collection of a-tag elements
            for (var i = 0; i < els.length; i++){

                    //attempt to find a data-preview attribute node
```

```
                var att = els[i].getAttributeNode("data-preview");
                //if it's there
                if(att != null){
                        //add it to the array
                        links.push(els[i]);
                }
        }
        //return the image-linking a-tag elements
        return links;
    }

    function showPreview(url){

                //creates a pop up div that

                        //disables the controls on the page

                        //creates and displays an image previewer element

                        //loads the given image into the preview

                        //and creates a close button which is assign its own event handler
that

                                //unloads the images by hiding the div and enabling the
controls on the page

    }

    return Preview;
})();
```

This shows how blocking and pseudo coding can let me quickly work through the problems that I know how to solve, and set aside those problems that will require some extra thought until later. In this case I'm not sure how I want to handle the display. In the end I decide that rather than creating the display elements dynamically, as I wrote in the original pseudo code, I want to display the image overlays by simply manipulating CSS styles to show and hide elements that were already in the HTML. The final script looks like this.

Fourth Snapshot:

```
//public gloabal variable, Preview, holds the module to manage the image preview
var Preview = (function(){

    //literal object that's returned and assigned to the global variable of the same name
    var Preview = {};

    //an array to hold the image links, indexed by the data-preview attribute
    var images = [];

    //the one public method for this Preview API
    Preview.init = function(){

        //get all the a tags and collect all those with the attribute "data-preview"
        var links = getImageLinks();

        //loop through the collected links and assign an onclick function handler that
```

```
            for (var i = 0; i < links.length; i++){

                    //add the image url to the images array
                    images[links[i].getAttribute("data-preview")] =
links[i].getAttribute("href");

                    //assign an event handler to the a-tag element's onlick event
                    links[i].onclick = function(e){

                            //cancels the default action
                            if(e.preventDefault){
                                    e.preventDefault();
                            }
                            //preview the image associated with this link
                            showPreview(this.getAttribute("data-preview"));

                            //return false to cancel the default event (for browsers that don't
implement the prevent default method)
                            return false;
                    };
            }

            //assign event handler to overlay "close" icon
            document.getElementById("preview-close").onclick = function(){
                    document.getElementById("preview-image").src = "";
                    document.getElementById("preview").style.display = "none";
            };
    };

    //function retrieves all of the page links that have images to preview.
    function getImageLinks(){

            var els = document.getElementsByTagName("a"), //a collection of all the a a-tag
elements on the page
                    links = []; //an array to hold all the a-tag elements that link to images

            //loop through the collection of a-tag elements
            for (var i = 0; i < els.length; i++){

                    //attempt to find a data-preview attribute node
                    var att = els[i].getAttributeNode("data-preview");
                    //if it's there
                    if(att !== null){
                            //add it to the array
                            links.push(els[i]);
                    }
            }
            //return the image-linking a-tag elements
            return links;
    }

    //sets the styles to show the preview with the selected images
    function showPreview(ix){
            document.getElementById("preview-image").src = images[ix];
            document.getElementById("preview").style.display = "block";
    }

    return Preview;
})();
```

4. CODE DEVELOPMENT

As you have seen, in this approach to app development, code development overlaps with code blocking and pseudo-pseudo coding, so there's really not much more to say here, other than this. All of these tasks are intended to be used recursively. At points along the way you may find that something was overlooked, or you may realize that there's a better way to accomplish specific tasks. In any of these cases, you may need to return to your problem analysis tasks, or you may want to edit your models, or may find a better way to organize your code, so it's important to be familiar and comfortable with each technique.

5. TESTING

The subject of testing is a big topic. Modern JavaScript developers use an automated process know as Unit Testing, but you can do a pretty good job by manually testing your code. Test your inputs thoroughly. Enter blank values or wrong data types and see how your app responds. Test your app on multiple browsers, in different versions if possible. It may be difficult to be rough with something you artfully created, but you need to try to break your app. Be merciless, because your end users will. It is better for you to find the flaw in your app than for someone else to point it out. I've had both experiences and the second is way worse.

Sean Olson
Riverside JS Workshop
RiversideJS.net
4/5/2015