# Software Design

The overall design is broken into two parts: the UI and the motor controls. The separation was designed to allow coding before the physical prototype was complete, simplify each component and reduce errors.

## UI

The UI consists of several functions. The most important function, the UIListSelection function is passed an array of strings. The function takes input from the user and updates the screen, returning the index of the selection. The function is called twice, once for choosing the drink type and again for choosing the drink size. The program will then calculate the amount of time each valve must be open, then open and close the appropriate valves in the background, while simultaneously updating an onscreen animation of a cup filling. The program will then return to the first screen.

Dirty Mutt   
Coke   
….

O Splash

Drink Levels: a b c

2oz 60mL

Half Cup 125mL

…

Bottle 591mL

Drink Levels: a b c

Drink Levels: a b c

## Motor Controls

The background consists of several functions to control the motors. There are two functions that open and close valves, with 6 task that operate as wrappers. This allows the calling code to open and close valves in the background, as well as creating an abstraction layer.

## Important Functions and Blocks

### Ratio Calculator: ratioCalc

*Parameters: volume, three ratios, three times (ref).*

*Return: void*

Calculates the volume of each liquid required. Then calculates the time (s) each valve is open for.

Coder: Alex

### Open Valve: openValve

*Parameters: motor, number of attempts*

*Return: void*

Opens valve by running motor. When the motors stops moving, the motor retreats and restarts moving. This process repeats for the number of attempts. This is due to physical constraints of the materials

Coder: Glenn

### Close Valve: closeValve

*Parameters: motor, number of attempts*

*Return: void*

Closes valve by running motor. When the motors stops moving, the motor retreats and restarts moving. This process repeats for the number of attempts. This is due to physical constraints of the materials.

Coder: Glenn

### UI List Selection: UIListSelection

*Parameters: list of strings, size of list, number of lines.*

*Return: short*

Function will output the list of strings to the screen. The function will then wait for user input to either return the selection, move upward or downward through the list. If the number of strings exceeds the number of lines, the list will scroll with the user. If the user reaches the bounds of the list, the selection will loop to the other end.

This function will be called for selecting drinks and drink sizes.

Coder: Kiran

### Pour Animation: pourAnim

*Parameters: none*

*Return: task*

Function prints an animation of a cup filling on the screen. Since this is a task, no parameters can be passed. However, the animationTimeRequired variable is used to pass the amount of time required for the animation.  
Coder: Kiran and Alex

## Testing Procedures

All functions were divided into two categories: normal functions and hardware interacting functions.

The self-contained functions were tested using an entirely separate program before being implemented into code. These functions were called using both common and edge case parameters and checked to ensure they performed as expected. For example, the ratioCalc function was given the parameters of several drink and drink sizes (coke, dirty mutt, etc. for 125mL, 250mL, 355mL) and printed the corresponding times to the display. The outputs are then compared with values calculated by hand. If any of the values did not match, the function would be reanalyzed. Once the function passed all cases, it was implemented into the code and quickly checked to make sure everything operated roughly as expected.

For any function that interacted directly with hardware, a similar procedure was taken with regular functions. In addition, these functions were tested for reliability. For example, the program alternated calling the open valve and close valve functions. It was then observed if the mechanical valve both fully opened and closed each time, and that they were open the same amount (with roughly the same liquid flow rate). This was to ensure the hardware operated consistently.

## Significant Problems

One significant problem faced was RobotC’s inability to pass arrays of strings. This was a critical feature required for creating reusable user interfaces. To solve this problem, functions were able to take a pointer to a string as an argument. Inside the functions, the pointer was augmented before being dereferenced. This is an excerpt from the UIPrint function.

for (short i = 0; i < nLines; i++)

{

nxtDisplayClearTextLine(i);

// Using the pointer like an array

nxtDisplayString(i, " %s", **\*(list + i)**);

}

Another major problem faced was RobotC’s inability to pass parameters to tasks. This design decision make sense for inexperienced programmers, as passing local variables by reference may have unintended consequences if the calling stack frame gets popped (which may happen). To solve this limitation, a minimal number of global variables were used with maximum caution. There was no way of avoiding them.