Progress Report: Design Updates

Kevin Juan and Sofya Calvin

May 15, 2016

1 Introduction

1.a AguaClara Overview

AguaClara is an engineering student project team at Cornell University started in 2005 with the goal of bringing sustainable water treatment to the Global South. Through connections with local communities, AguaClara has been able to implement many economic, gravity-powered water treatment plants in Honduras. Members on the team carry out the research to improve the technologies and help generate designs for treatment plants and components. All of this is made possible through the communication between engineers at the water treatment sites and students on campus, which the design team helps facilitate.

1.b Design Team Overview

The Design Team works on various projects which involve automating the design process for AguaClara's water treatment technology. Designs are coded using MathCAD and drawn out via AutoCAD and then published on the AguaClara server for partners to use. The overarching aim of this team is to allow users to enter various parameters about a plant and be presented with detailed AutoCAD designs, a materials list, and a report describing any information needed to build the technology.

2 Modular Design Challenge Details

The Modular Design Challenge for Spring 2016 was to continue the progress made in the Fall 2015 term to automate designs for individual components of AguaClara water treatment plants. Fall 2015 saw the continuation of Heidi Rausch's work in 2012 by Stephanie Sun to update pre-existing component files and create new files for components not previously available. This spring, the team checked that the component designs were up-to-date with current technologies used by AguaClara, found solutions to errors identified in the fall, and created files for components not yet available on the beta server. The beta server is a server built specifically to test how designs will draw when committed to the actual server that partners in Honduras have access to. When the design team finishes a piece of code, they commit it to the beta server to make sure that it draws out exactly as it should when on the actual AguaClara server.

2.a OStaRS/EStaRS

When the challenge was first taken on, there was no previously written modular code for the Stacked Rapid Sand Filter. Furthermore, although there was code already written for both the OStaRS and EStaRS filters within the entire plant, there were no files to combining the two filters into one code. Therefore, one of the biggest challenges in this task was finding a way to combine the two codes into one functional modular code that could distinguish between when the flow rate constituted an OStaRS or EStaRS filter.

The first step to writing this code was investigating the EtFlocSedFi file in the design folder. Browsing through the entire plant code with an OStaRS filter, the team identified which references and variable inputs would be needed to draw only the filter. After acquiring all of the references and formulas needed to draw an OStaRS filter, the team wrote and tested a modular code to draw an OStaRS filter. Once that was

debugged, the process was repeated using the EtFlocSedFiLow file to write the modular code for EStaRS, the enclosed flow filter.

The final step to this project was to take the two, separate modular codes and combine them. This task would have been relatively simple if both of the individual codes didn't refer to and redefine the same variables. Because this was the case, the team had to find a way to draw both filters for any given set of input variables but only output the code for one of them. To do this,the output variables (which code for the drawings on AutoCAD) had to be redefined for OStaRS and EStaRS. These new variables could then be incorporated into an if-statement, as shown in Figure 1, to determine which filter would actually be used.

$$AC_{Filter} := \begin{cases} AC_{FilterLow} & \text{if } Q_{Plant} \le Q_{PlantMaxLF} \\ AC_{FilterHigh} & \text{otherwise} \end{cases}$$

Figure 1: If-statement that determines whether an OStaRS or EStaRS filter is drawn based on the flow rate.

This simple if–statement solved the problem as it essentially assigns the final variable, which codes for the appropriate filter, based on whether or not the flow rate is less than or greater than the predefined variable Q.PlantMaxLF. This variable is defined in the expert inputs file and represents the transition flow rate between OStaRS and EStaRS. When the flow is less than or equal to that value, the AC.Filter codes for an EStaRS low flow filter; when the flow is greater than that value, AC.Filter codes for an OStaRS high flow filter.

After the if-statement was inserted, the code was tested on the beta server to see if it would properly draw the correct filter. Testing on the beta server found several flaws in the code, all of which were resolved. The first issue found was that the beta server would not display any variables in the About.html link other than the ones that the user defined or stock variables. The fix for this was to add a section to the code that explicitly showed the variables to be displayed. However, this was not enough as both EStaRS and OStaRS had overlapping variable names, causing some variables to be incorrect. The solution, proposed and implemented by Meghan Furton, was to filter through the variables for EStaRS, add "Low" to the variable name, and then insert them into the display variables section.

The other major flaw that the beta server revealed was that AutoCAD designs were not being produced for flow rates less than 5 L/s. Running the code locally in the erroneous range of values found that the code for OStaRS was not designed to produce viable code for flow rates below 5 L/s. Because of this, AC.FilterHigh was always undefined, which would create a broken if-statement that would not produce an output for AC.Filter. The work around for this issue was to create a new variable Q.PlantKeep. This variable would be assigned the value that the user gave to Q.Plant. If Q.PlantKeep was greater than 5 L/s, then Q.Plant receives the same value that was put in. Otherwise, if Q.PlantKeep was less than 5 L/s, then Q.Plant was assigned an arbitrary flow rate above 5 L/s to prevent AC.FilterHigh from being undefined. For the code, the arbitrary value was set to 20 L/s. The code for redefining Q.Plant occurred before the code that produced AC.FilterHigh, and looked as such:

$$Q_{PlantKeep} := Q_{Plant}$$

$$Q_{Plant} := \begin{cases} Q_{PlantKeep} & \text{if } Q_{PlantKeep} > 5\,\frac{L}{s} \\ 20\,\frac{L}{s} & \text{otherwise} \end{cases}$$

SRSF High

Figure 2: Code that reassigns Q.Plant.

After AC.FilterHigh was defined by either the initial or arbitrary value, Q.Plant was redefined as Q.PlantKeep, the user input value, prior to the SRSF Low code seen in Figure 3. This redefinition was

critical to the proper functioning of AC.Filter because if the starting flow rate was less than 5 L/s, Q.Plant would take on 20 L/s. However, this was only needed to make sure AC.FilterHigh was defined. Restoring Q.Plant to its original value would ensure that the flow rate for EStaRS calculations was correct. This safety measure was only necessary for flow rates lower than 5 L/s as any value between 5 L/s and the threshold flow rate would still draw the proper EStaRS.

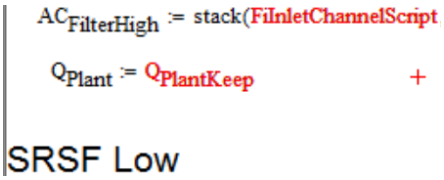

Figure 3: Q.Plant being redefined before the SRSF Low code.

Once all of the corrections were made to the code and all of the bugs were removed, the beta server was able to draw and display the correct sand filters along with all of the specified variables. In addition, no ambiguities were found in drawing the filter at the threshold flow rate.

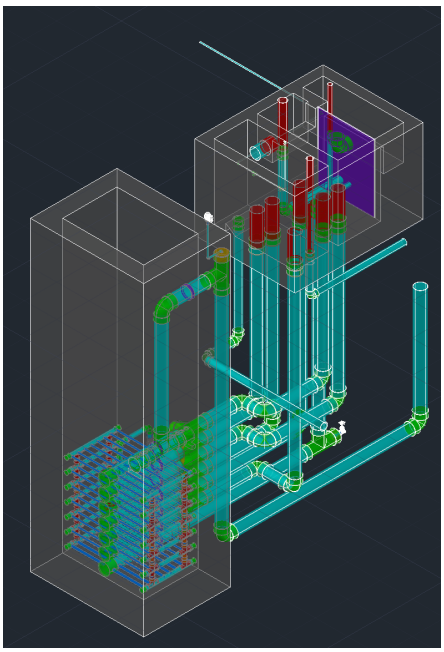

Figure 4: The AutoCAD drawing of a 10 L/s OStaRS using the modular code.

2.b Chemical Dose Controller and Chemical Storage Tanks

Much like EStaRS and OStaRS, the chemical dose controller and chemical storage tanks also lacked modular code. The goal for the modular code of the CDC was to write it such that it would draw both the CDC as well as the chemical storage tanks. The major challenge of this was taking two different AC functions, AC.Cdc and AC.ChemTanks, and combine them into one AC variable that would be named AC.CDCChemTanks.

The process of writing the modular code was much like that of the combined EStaRS and OStaRS code where the first part was to investigate the EtFlocSedFi code for the references and variables directly and indirectly needed to draw the CDC and chemical storage tanks. Once all of the correct variables and references were found, the challenge of compiling both drawings into one was taken on. The solution to this challenge was relatively simple as a new AC variable called AC.CDCChemTanks was defined. The new variable took both the outputs from AC.Cdc and AC.ChemTanks and stacked it into one. In order for this

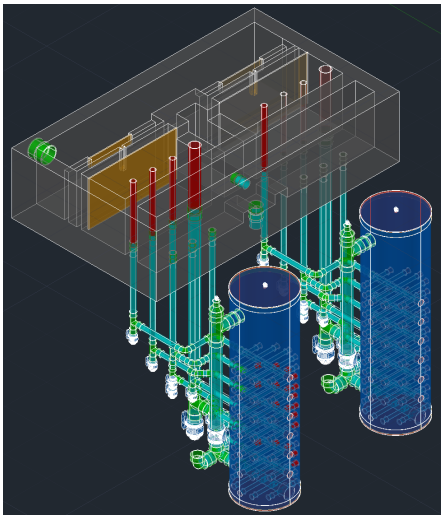

Figure 5: The AutoCAD drawing of a 5 L/s EStaRS using the modular code.

to work, the newly defined AC.CDCChemTanks variable had to be placed after the definitions for AC.Cdc and AC.ChemTanks. The reulting AC variable looked like:

$$AC_{CDCChemTanks} := stack(AC_{Cdc}, AC_{ChemTanks})$$

Figure 6: The newly defined AC.CDCChemTanks variable stacks the AC.Cdc and AC.ChemTanks variables.

When the code was originally tested on the beta server for the CDC and Chemical Storage Tanks, the code only created the tanks and the CDC as free floating objects. This was exactly what was desired as the modular code intended only to show the components without any of the masonry. However, the issue with the original code was that the CDC and Chemical Storage Tanks existed as separate, disconnected pieces.

The missing piece to the code and model was the script that implemented the designs for the plumbing and flexible tubing that would connect the two parts. Design Team member, Meghan, was tasked with writing the code for the plumbing and flexible tubing that connected the two parts. Once the script for the plumbing was finished, the variable ChemConnectionPluStack was added to the AC.ChemTanks variable.

$$AC_{ChemTanks} := stack(ChemTankScript, ChemCoagPlumbingScript, ChemChlorPlumbingScript, ChemTanksReflectRotateScript, ChemConnectionPluStack)$$

Figure 7: The new AC.ChemTanks variable includes the variable ChemConnectionPluStack that draws the piping from the Chemical Storage Tanks.

The resulting AutoCAD design drew the Chemical Storage Tanks with PVC pipping connected to it, but still had the CDC as a free floating unit. The lack of connection between the two units was due to the fact that the flexible tubing code was not finished, so it was not implemented into the code. Addition of the finished flexible tubing code was deemed as one of the final tasks to completion of the CDCChemTanks modular code.

An issue that occurred with the code was the improper rotation and drawing of the Chemical Storage Tanks. At large flow rates, namely above 50 L/s, the Chemical Storage Tanks were rotated such that the four tanks could fit within the masonry (not drawn in the model). The issue with the rotation was that while the tanks were rotated, parts of the script that were necessary to subtracting out pieces from the Chemical

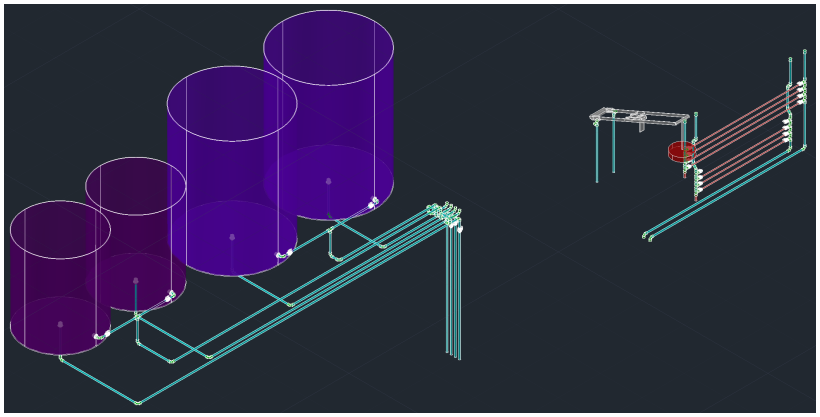

Figure 8: The AutoCAD model of the Chemical Storage Tanks and CDC with the piping only for 10 L/s plant flow rate.

Storage Tanks were left untouched. AutoCAD subtraction requires that the shape of the subtraction be drawn first as a solid before it is taken out of the desired piece. The problem with the rotation was that the subtracted pieces' positions were not being changed, causing AutoCAD to draw the solid and not subtract it from the Chemical Storage Tanks, which left cylinders and boxes in the model space.

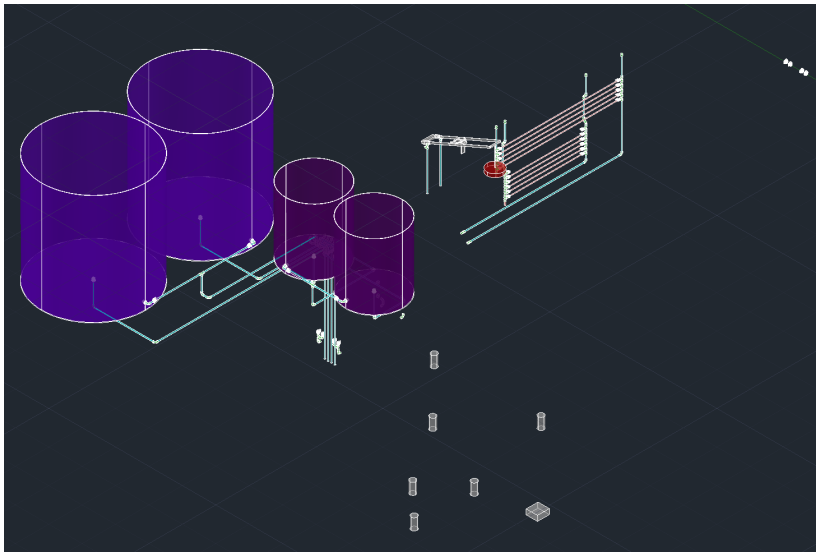

Figure 9: Cylinders and boxes that appeared from the rotation.

The error in the cylinders and boxes being drawn was traced back to the variable ChemTanksReflectRotateScript, which was a component of AC.ChemTanks. The problem was linked to the thawing and freezing of certain layers when the model was being generated by AutoCAD. When the solids were drawn in the wrong place, a new layer was created for them. After AutoCAD failed to subtract the pieces correctly, the solids were frozen in their layer, which left them in the model space. To fix this, the subtracted pieces had to be thawed after they were frozen so that they could be rotated to the correct spot. Once they were correctly placed in the model, the subtracted pieces layers could be frozen again. By implementing the redefinition code in Figure 10, the cylinders and boxes present in Figure 9 were successfully removed for CDC and Chemical Storage Tank designs for plant flow rates greater than 50 L/s.

To finish the design for the CDCChemTanks, the rest of the plumbing and flexible tubing from Meghan's challenge had to be implemented. For this task, there was nothing to modify within the CDCChemTanks modular code as all of the changes took place within files that were referenced by the modular file. Because of

```
ChemTankPositionScript := | stack(ThawChlorPieces, RotateChemPieces, SlideChemPiecesY, FreezeChlorinePieces)  if  L_ChemTanksTotal > W_ChemPlatform
                          | stack("")   otherwise


ChemTanksReflectRotateScript := stack(ChemReflectScript, ChemTankPositionScript)
```

Figure 10: Modifications to the position variable.

this, all of the additions would automatically apply. The first round of testing the tubing and piping changes on the CDCChemTanks modular code found issues with the flexible tubing when the beta server produced the design. To understand the issues and solutions better, it is recommended that Meghan's progress report is consulted with.

Once the kinks in the flexible tubing were fixed, the beta server successfully produced the fully integrated CDCChemTanks modular design. Figure 11 is an example of the final design produced by the beta server.

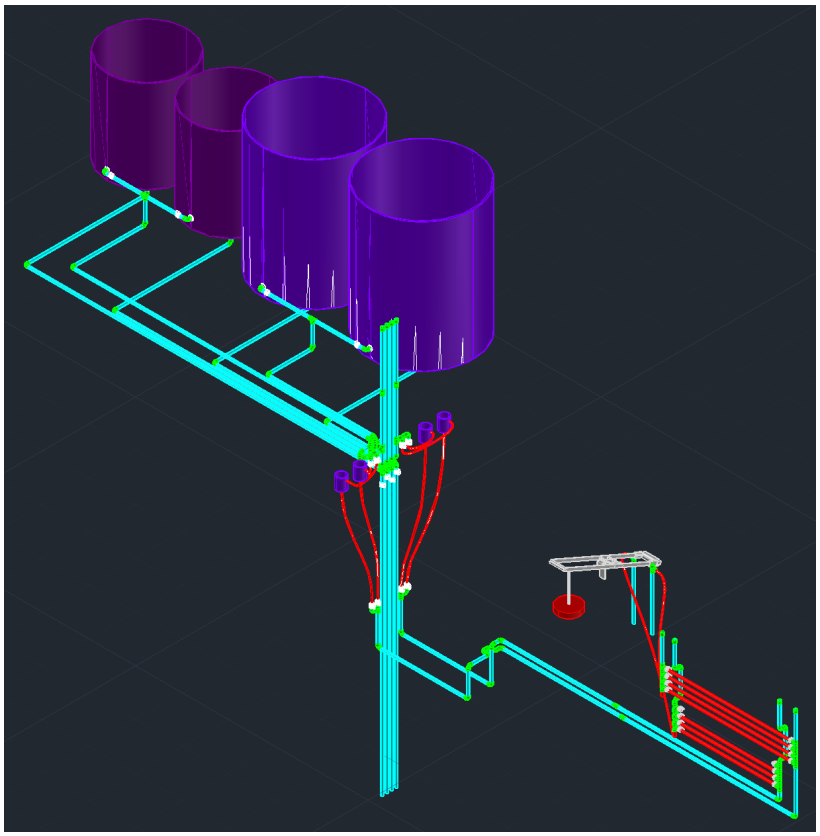

Figure 11: The CDC and Chemical Storage Tanks connected by piping and flexible tubing.

2.c LFOM and Orifice Template

The Linear Flow Orifice Meter (LFOM) is a short straight section of pipe which measures the water flow through the plant and creates a linear relationship between the depth of water in the entrance tank and the flow rate through the plant. The LFOM is unique because because it has many orifices, or holes, in the pipe wall. The number and placement of the orifices is dependent on the plant flow rate. When the team first took on this portion of the modular challenge, the code for the LFOM was defined within the large scale plant and there was a second code to draw out the orifice template. The orifice template code consisted

of orifices, with cross hairs centered in each, all within the boundaries of sheets of paper. The overarching goal was to create a modular code that not only drew the LFOM at various flow rates, but one that also produced an attached file with the appropriate orifice template.

As the team set out to create this code, it seemed like a simple task. All that had to be done was combine the code to produce the LFOM with the code to produce the template and then create a final output variable which stacked the two parts of the code. However, after committing this to the beta server, it became clear that there were many problems, specifically with the code for the orifice template. However, the code for the orifice template that was left in the Mathcad document set a solid foundation for achieving the overall objective, but still contained many flaws. Because of this, it was sensible reconstruct and revise the current code to be simpler and functioning rather than start from scratch.

The first error encountered in the code was that the template was not producing the proper number of orifices at various flow rates. After looking over the code and testing various variables, it was found that the server had trouble drawing both the LFOM and the template at once - perhaps because of an overlap of variables within each code. Therefore, the team decided to create two duplicate Mathcad files with all of the LFOM code and references needed. One file was responsible for generating a model of the LFOM by returning only AC.Lfom. The other file was responsible for the template by returning only AC.LfomTemplate. This resulted in the creation of two different modular codes that performed the two different tasks assigned to the files. This ultimately corrected the number of orifices drawn.

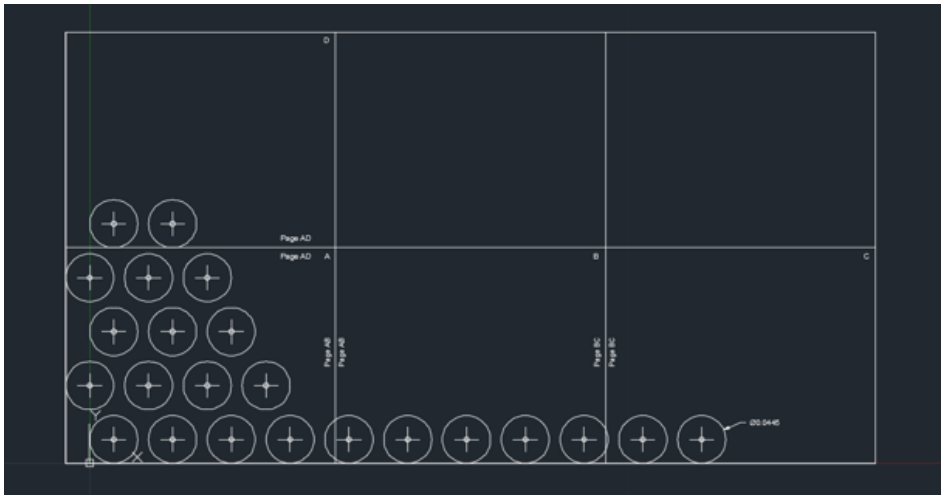

Figure 12: Orifice template at 20 liters per second using the original code.

The second error encountered also involved the template code. Even though the number and placement of orifices on the template was correct, the pages themselves were presenting many issues. For some flow rates, the top boundary of the papers were elevated and disconnected to produce a template similar to Figure 14. For other flow rates, the template was not drawing the proper number of papers to fit the amount of orifices so some of the orifices were being cut off by the paper limitations, which is later shown by Figure 24. To fix these issues with the paper, the team looked into how the boundaries of the paper were defined. As it turned out, the code for the boundaries utilized the floor function to round calculated values down. The team tried changing this to the ceil function (to round calculations up) which proved to be an effective solution for flow rates where orifices were cut off. However, this created larger issues with the template. The issue with the floating upper boundary still persisted, but for all flow rates, the paper layout view was cutting off at the wrong points. This resulted in pages not being cutoff at the specified borders drawn into the AutoCAD drawing as shown in Figure 13

Yet another issue with the original template code present was that the template would be improperly scaled, resulting in the pages defined in the model space to appear on the paper space as a tiny rectangle. The issue that was causing this was that the paper space in AutoCAD defaulted to 11 by 8.5 in no particular set standard of units. Because of this, if calculations were performed in meters as they were, the paper space would automatically take on units in meters. Likewise, if calculations were done in inches, the paper space

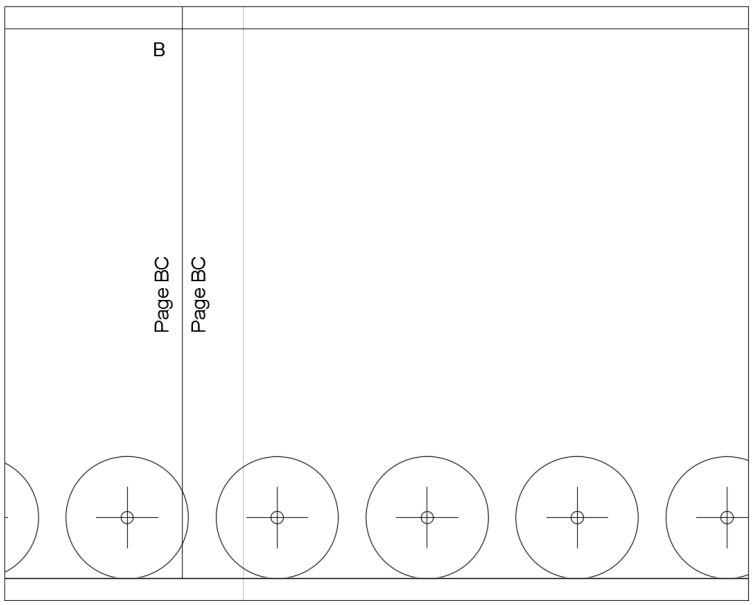

Figure 13: Layout view of an improperly cut off page.

would take on inches. AutoCAD would never convert units into on standard, causing us to end up with a layout that was approximately 0.2 m x 0.2 m plotted on a paper that was 11 m x 8.5 m, and likewise for inches. These always led to an extremely small layout on a huge blank paper space as shown in Figure 15.

All of these issues with the template drawing led the team to critically reevaluate the effectiveness of the code that defined the templates and page layouts, and then create new code and edit old code to function better. The first change made to the code was to redefine how the length and width of the border of each page were calculated. The original code attempted to define the length and width of the border using complicated floor functions on different orifice variables like the diameter, orifice height from the x-axis, and the length of one orifice diameter and the spacing between orifices as shown in Figure 16. Since changing the floor function to a ceiling function was an ineffective solution, the calculations were completely redefined. To redo the calculations, the team utilized the fact that AutoCAD's paper space creates a dashed outline seen in Figure 14 that sets the printable area. Anything outside of the dashed border would not be printed by AutoCAD. For practicality, the size of the paper space chosen was standard 11" x 8.5" letter paper, and is the default paper space set by AutoCAD. On this paper space, AutoCAD automatically sets a 0.25" margin on each side, meaning the printable area was only 10.5" x 8". The team used these dimensions to make the length and width of the border fit exactly within the printable area, which led the length and width of the border to be calculated by subtracting the total size of the margin from the paper length and width. This ultimately simplified the border dimensions by making them the exact same dimensions of the printable area. Figure 17 shows the new code.

The next part of the script was to figure out how many horizontal, vertical, and transverse layouts would be needed to capture all of the LFOM orifices on the paper space. Much like the rest of the old code, the parts used to define the number of horizontal, vertical, and transverse layouts were overly complicated, involving if statements and ceiling and round functions. Figure 18 shows the complexity of the old code when determining the number of layouts to produce. Rather than making the code conditional, the team used the fact that each layout would be exactly the dimensions of the printable area to our advantage to calculate the number of each type of layout.

To calculate the number of horizontal layouts, N.HLayouts, the total length of the template had to be known. One of the key facts about the total length is that it is dependent on the row with the most number of orifices. To find the maximum number of orifices in one row, the function max was used on the N.LfomOrifices array to return the maximum number of orifices in any row. With this value, the total length of the template, L.Temp, would be the sum of the diameters of all the orifices and the spacing between all of the orifices. Figure 19 shows the the template length calculations. The spacing length was multiplied by

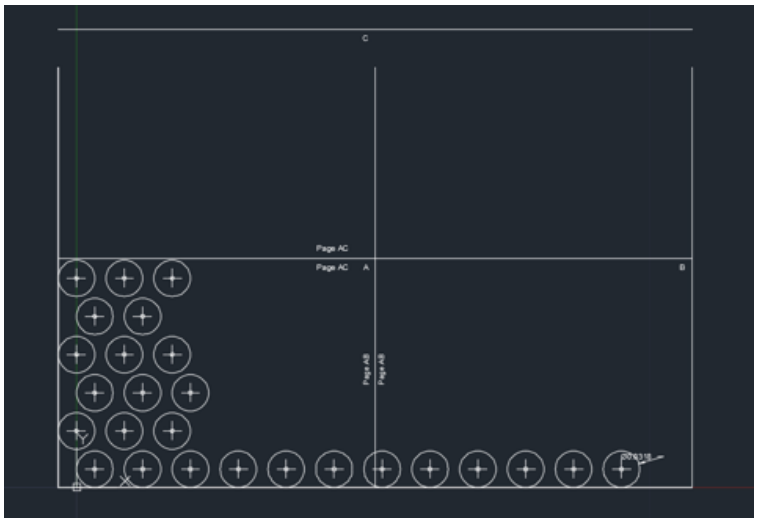

Figure 14: Orifice template with error in paper boundaries.

Figure 15: Improperly scaled template on the paper space (Bottom Left Corner).

$$L_{Border} := \text{floor}\left(\frac{L_{Paper}}{OrificeCenterDist}\right) \cdot OrificeCenterDist - D_{LfromOrifices}$$

$$W_{Border} := \text{floor}\left(\frac{W_{Paper}}{H_{LfromOrifices_1} - H_{LfromOrifices_0}}\right) \cdot \left(H_{LfromOrifices_1} - H_{LfromOrifices_0}\right)$$

Figure 16: Original code that defined the border length and border width.

$$L_{Paper} := 11in$$

$$W_{Paper} := 8.5in$$

$$L_{Border} := L_{Paper} - 0.5in$$

$$W_{Border} := W_{Paper} - 0.5in$$

Figure 17: New code that defined the border length and border width as the dimensions of the printable area.

$$N_{HLayouts} := \left| \begin{array}{l} Layouts \leftarrow round\left[\frac{\left(OrificeCenterDist \cdot N_{LfromOrifices_0} + D_{LfromOrifices}\right)}{\left(L_{Border}\right)}\right] \text{ if } \frac{\left(OrificeCenterDist \cdot N_{LfromOrifices_0} + D_{LfromOrifices}\right)}{\left(L_{Border}\right)} > 1.5 \\ Layouts \leftarrow ceil\left[\frac{\left(OrificeCenterDist \cdot N_{LfromOrifices_0} + D_{LfromOrifices}\right)}{\left(L_{Border}\right)}\right] \text{ if } \frac{\left(OrificeCenterDist \cdot N_{LfromOrifices_0} + D_{LfromOrifices}\right)}{\left(L_{Border}\right)} < 1.5 \\ return\ Layouts \end{array} \right.$$

$$N_{VLayouts} := round\left(\frac{H_{LfromOrifices_{last(H_{LfromOrifices})}} + \frac{D_{LfromOrifices}}{2}}{W_{Border}}\right)$$

$$N_{TLayouts} := \left| \begin{array}{l} Layouts \leftarrow round\left(\frac{OrificeCenterDist \cdot N_{LfromOrifices_{last(N_{LfromOrifices})}} + D_{LfromOrifices}}{L_{Border}}\right) \text{ if } \frac{OrificeCenterDist \cdot N_{LfromOrifices_{last(N_{LfromOrifices})}} + D_{LfromOrifices}}{L_{Border}} > 1 \\ Layouts \leftarrow ceil\left(\frac{OrificeCenterDist \cdot N_{LfromOrifices_{last(N_{LfromOrifices})}} + D_{LfromOrifices}}{L_{Border}}\right) \text{ if } \frac{OrificeCenterDist \cdot N_{LfromOrifices_{last(N_{LfromOrifices})}} + D_{LfromOrifices}}{L_{Border}} < 1 \\ return\ Layouts \end{array} \right.$$

Figure 18: Old code that calculated the number of layouts.

$$L_{Temp} := \max(N_{LfomOrifices}) \cdot D_{LfomOrifices} + (\max(N_{LfomOrifices}) - 1) \cdot S_{LfomOrificesMin}$$

$$H_{Temp} := H_{LfomOrifices_{last(H_{LfomOrifices})}} + \frac{D_{LfomOrifices}}{2}$$

Figure 19: Code that calculates the entire template length and width.

$$N_{HLayouts} := ceil\left(\frac{L_{Temp}}{L_{Border}}\right)$$

$$N_{VLayouts} := \begin{cases} 0 \text{ if } \frac{H_{Temp}}{W_{Border}} \le 1 \\ ceil\left(\frac{H_{Temp}}{W_{Border}}\right) \text{ if } \frac{H_{Temp}}{W_{Border}} > 1 \end{cases}$$

$$N_{TLayouts} := \begin{cases} 0 \text{ if } N_{VLayouts} = 0 \\ N_{HLayouts} \text{ if } N_{VLayouts} \neq 0 \end{cases}$$

Figure 20: New code that calculates the number of horizontal, vertical, and transverse layouts.

the max of N.LfomOrifices minus one because the number of spaces between orifices is one fewer than the number of orifices. Using the total length of the template, the number of horizontal layouts needed could be determined by taking the ceiling of L.Temp divided by L.Border. Dividing L.Temp by L.Border would produce a number that indicated how many borders of length L.Border could fit the length of the template. Taking the ceiling of this value ensures that the number was always rounded up to the nearest integer so as not to miss any orifices and because only whole numbers of pages are desired. The first line in Figure 20 shows the new code for N.HLayouts.

Calculating the number of vertical layouts was similar and different from the number of horizontal layouts. The first thing that needed to be done was figure out the height of the template. The variable H.Temp in Figure 19 shows how the template height was calculated as the height of the centers of the orifices in top row plus an orifice radius.

Unlike the calculation for the number of horizontal layouts, the calculations for the number of vertical layouts needed was conditional. The first statement was that if the ratio between H.Temp and W.Border was less than or equal to one, then the number of vertical layouts defaulted to 0. This was implemented because if that ratio was less than or equal to one, then the entire layout can vertically fit on one page. Because N.VLayouts calculates how many rows of layouts are needed above the row of horizontal layouts, if the ratio between H.Temp and W.Border was less than or equal to one, then adding an extra row would produce redundant blank templates.

If the ratio between H.Temp and W.Border exceeded one, then the number of vertical templates would be calculated as the ceiling of that ratio. Ceiling the ratio was important for this condition because then the number of vertical layouts would be rounded up to the nearest whole number. This would then create the proper whole number of extra rows needed to fit the template vertically. Figure 20 shows how this was implemented.

As for the number of transverse layouts, the new code also greatly simplified the calculations. The number of transverse layouts is defined as the number of horizontal layouts that don't occur in the bottom row of layouts. Because of this fact, the number of transverse layouts could simply be equated to the number of horizontal layouts only if there were extra rows needed as determined by N.VLayouts. Otherwise, there would be no transverse layouts since N.VLayouts was zero, meaning no extra rows were needed. The old code produced code that was going along that idea, but was never implemented correctly. This led to layouts

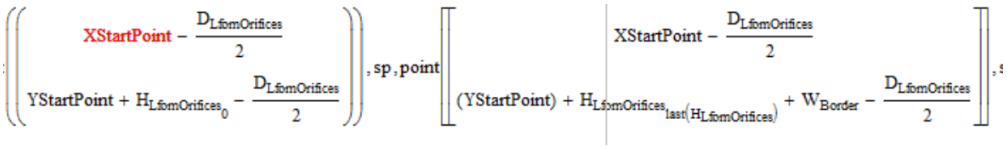

Figure 21: Old code that defined the start and end points for the vertical boundary.

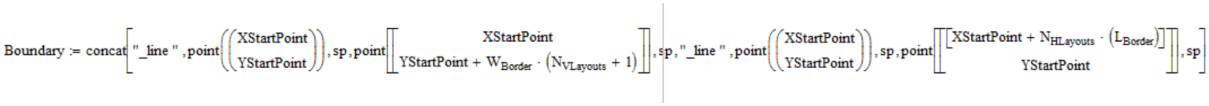

Figure 22: New boundary code. The vertical boundary points are the first two points while the horizontal boundary points are the second set.

not in the first row to be unlabeled, which was mostly an aesthetic issue. Figure 20 shows how the number of transverse layouts was determined.

After modifying the calculations for the layout border dimensions and the calculations for the number of layouts, the next part that had to be fixed was the issue with drawing the physical boundaries. This amounted to changing the definitions of the Boundary variable and implementing these changes into the DelineateL and DelineateW variables. One of the issues with the old boundary code had to do with selecting the right points for drawing a vertical line. The old code correctly selected the first point by making the x-coordinate the leftmost point on the template and the y-coordinate be on the y-axis. However, the second point that defined the vertical line was consistently off because it didn't factor in the number of vertical layouts produced. Instead, set the end point to where the topmost orifice ended as seen in Figure 21. With the new code, the vertical boundary end point was changed to the sum of the vertical border lengths, which was the number of vertical layouts plus one times the width of the border. The reason that one had to be added to the number of vertical layouts was because N.VLayouts only counts the number of vertical layouts needed above the first row, so to account for all rows, one had to be added. This provided the fix to the issue with the border not being correctly drawn.

For the drawing of the horizontal border, the original code was sufficient for how it was written before, but had to be updated to match the new code. This was simply changed by making the first point for the horizontal line be the XStartPoint and the end point be the number of horizontal layouts times the border length. The reason that the first point was now XStartPoint rather than XStartPoint minus an orifice radius was due to a change that will be later explained.

Once the boundary line code was modified, the DelineateL and DelineateW variables were changed to adopt the boundary modifications. The DelineateL variable, which draws horizontal boundaries at the correct width points, now took on y-coordinates identical to the first two y-coordinates in the boundary code. The DelineateW variable, which draws the vertical boundaries at the correct length points, was changed to take on the x-coordinates of the second pair of points in the boundary code. Figure 23 shows the delineation code with the boundary modifications implemented.

After the boundary drawing code was fixed, the next step was to figure out how to scale the template correctly to fit on the paper space layout. The root of the cause for the template being plotted as a tiny corner on a large piece of paper was how the AutoCAD view port was set up for each different type of layout.

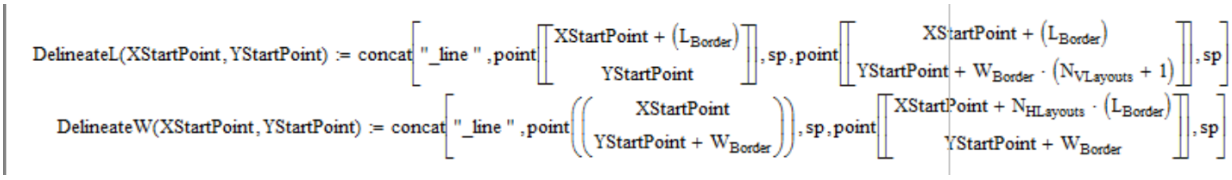

Figure 23: Code that copies the horizontal boundary and puts it a border width higher than the last one, and copies the vertical boundary places it a border length to the right of the last one.

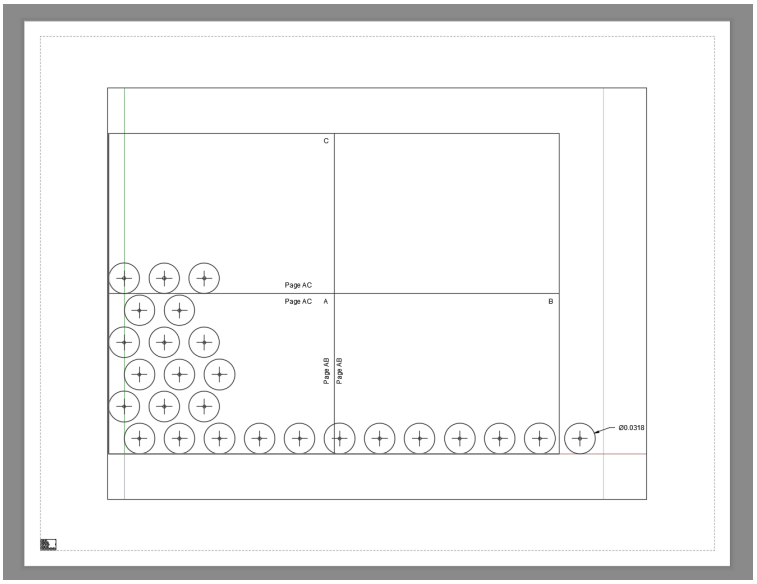

Figure 24: Paper space when the cleared space was changed to units of inches. The model space was not cleared out since the space to be cleared was in inches.

The original code started by clearing out a 10 m x 7.7 m blank space. It would then start the new view port at the point (0.013 m, 0.013 m) and extend it so that the area dimensions of the view port were the border length plus an orifice radius in length and the border width for the view port width. The first obvious issue with this is that the cleared out space was 10 m by 7.7 m, which would be much too large for the template, which was roughly 0.2 m by 0.2 m. The first approach to fixing this was to simply change the cleared area dimensions to inches. However, this caused a very small space to be cleared for the template to be printed, and didn't clear out the model space as seen in Figure 24.

Changing the space to be cleared for the view port to inches showed that AutoCAD was not converting units where they should have been. This meant the paper being drawn on was could have had dimensions in meters or inches. The fix to this was to scale the view port points in meters to correspond with the inches measurement. This was not the same as converting between units, but rather making everything such that 1 m was equivalent to 1 inch. This would mean that all of the calculations and points would be entered as meters and would remain as meters throughout the rest of the code, but when printed on the 11" x 8.5" paper, each meter would be 1 inch. To implement this, the right side point chosen to define the view port was multiplied by the unit-less conversion factor between meters and inches (39.37 in/m). This scaled everything such that 1 m equaled 1 inch on the paper. The final product would ultimately be printed in inches, and not meters as the Mathcad calculations would suggest. Figure 25 shows the old code without the scale factor applied. Figure 26 shows the scale factor being applied to the right side point, where L.Border is the x-coordinate and W.Border is the y-coordinate (The different points used between the view ports will be explained later).

After this scale was applied, AutoCAD began plotting the correctly sized layout on each sheet of paper. To confirm the results of the scaling factor, a template was physically printed on paper on a 1:1 scale and hand measurements were made. The results of hand measuring found that the actual printed template had the correct border dimensions, orifice diameter, and spacing as specified by the calculations done in Mathcad.

Following the changes made to scaling the templates, the last task that needed to be completed was to move the cluster of orifices from the far left of the template towards the middle as well as adjust the spacing between orifices on different rows. This modification took place in the section of code that drew the cross hairs and the circles of the template. In the original code, the section that determined the origins of the circles was not conditional. The code simply shifted the x-coordinate of the orifice origin to the right by an orifice radius if it was in an odd row, and left it at XStartPoint if it was in an even row.

Unlike other sections of the code, the revised section for determining orifice origin was made more complex

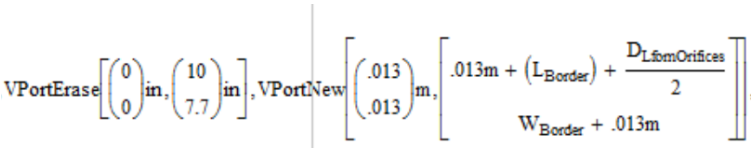

Figure 25: Old code that cleared the paper space and defined the new view port. In view port erase, the units were switched from meters to inches for testing.

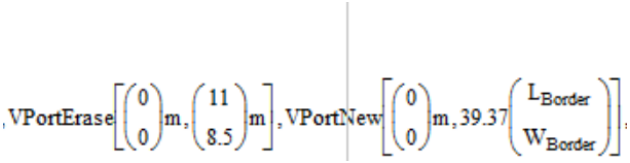

Figure 26: New code that cleared the paper space and defined the new view port with the proper scaling factor between meters and inches (39.37 in/m).

in order to produce an aesthetically correct template. The new code made the determination of the origin conditional, resulting in three unique calculations for the orifice origin. The condition that was used was again dependent on which row the orifices were in, h, where the 0th row was the bottom going to the top row, last(H.LfomOrifices).Though the same loop and condition was similar to the original code, the new code applied it in a different way. For the bottom row, h=0, the orifice origin was always shifted one radius to the right of the XStartPoint as shown in Figure 28.

For the rows above the bottom, the calculations got a little more complicated. The first thing that had to be done was create a temporary array with the number of orifices in each row, N.LfomOrificesTemp, but for the bottom row, the number of orifices was set to zero. Figure 29 shows how this was done. Once this was done, two different conditional statements were created. The first was for any row that was not the bottom row, and the number of orifices in the the bottom row was even (Second definition of origin in Figure 28). The other was for any row that was not the bottom row, and the number of orifices in the bottom row was odd (Third definition of origin in Figure 28). In both of these statements, the first shift made was to move the origin by a distance that was equal to half the length of OrificeCenterDist if the row was even, otherwise not for odd rows. This would ensure that the orifice origin of the current row lay dead center between two orifices from the row below it, which is critical to the structural integrity of the LFOM. The next shift done was to move each row by a distance of half the template length, which is the length of all of the orifice diameters and spaces in the bottom row. This movement made it so that the orifice furthest to the left aligned with the middle of the template.

After this point, the calculations were dependent on whether the number of orifices in the bottom row were even or odd. If the template was such that the bottom row had an even number of orifices, the orifices were shifted to the left by the product between OrificeCenterDist and the floor of half the maximum number of orifices not in the bottom row. This would ultimately align the middle of the cluster of orifices with the middle of the bottom row to achieve the proper aesthetic. Similarly, if the number of orifices in the bottom row was odd, then the only difference was to shift the cluster by the product of half the OrificeCenterDist and the floor of the half the maximum number of orifices not in the bottom row. This resulted in the

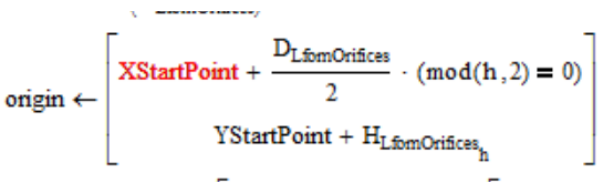

Figure 27: Old code that defined the origin based on whether the row it was in was an odd row or an even row.

$$\text{for } h \in 0..\text{last}\left(H_{LfromOrifices}\right)$$
$$\text{origin} \leftarrow \begin{pmatrix} XStartPoint + \frac{D_{LfromOrifices}}{2} \\ YStartPoint + H_{LfromOrifices_h} \end{pmatrix} \text{ if } h = 0$$
$$\text{origin} \leftarrow \begin{bmatrix} XStartPoint + \frac{OrificeCenterDist}{2} \cdot (\text{mod}(h,2) = 0) + \frac{L_{Temp}}{2} - OrificeCenterDist \cdot \text{floor}\left(\frac{\max\left(N_{LfromOrificesTemp}\right)}{2}\right) \\ YStartPoint + H_{LfromOrifices_h} \end{bmatrix} \text{ if } \left(h \neq 0 \wedge \text{mod}\left(N_{LfromOrifices_0}, 2\right) = 0\right)$$
$$\text{origin} \leftarrow \begin{bmatrix} XStartPoint + \frac{OrificeCenterDist}{2} \cdot (\text{mod}(h,2) = 0) + \frac{L_{Temp}}{2} - \frac{OrificeCenterDist}{2} \cdot \text{floor}\left(\frac{\max\left(N_{LfromOrificesTemp}\right)}{2}\right) \\ YStartPoint + H_{LfromOrifices_h} \end{bmatrix} \text{ if } \left(h \neq 0 \wedge \text{mod}\left(N_{LfromOrifices_0}, 2\right) \neq 0\right)$$

Figure 28: New code with three conditions to determine the placement of orifice origins.

$$N_{LfromOrificesTemp} := N_{LfromOrifices}$$
$$N_{LfromOrificesTemp_0} := 0$$

Figure 29: Creation of a temporary array that stores the number of orifices in each row, but with the bottom row set to have no orifices. This would ensure that the max of this array returned the maximum number of orifices in rows that were not the base.

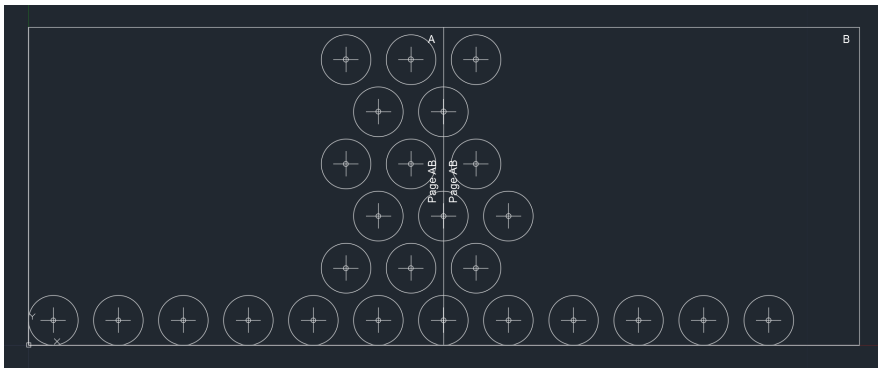

Figure 30: Model space view of a template for plant flow rate of 20 L/s.

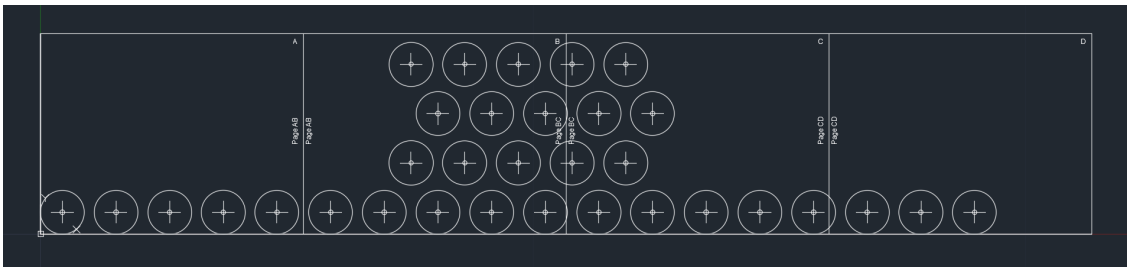

Figure 31: Model space view of a template for plant flow rate of 50 L/s.

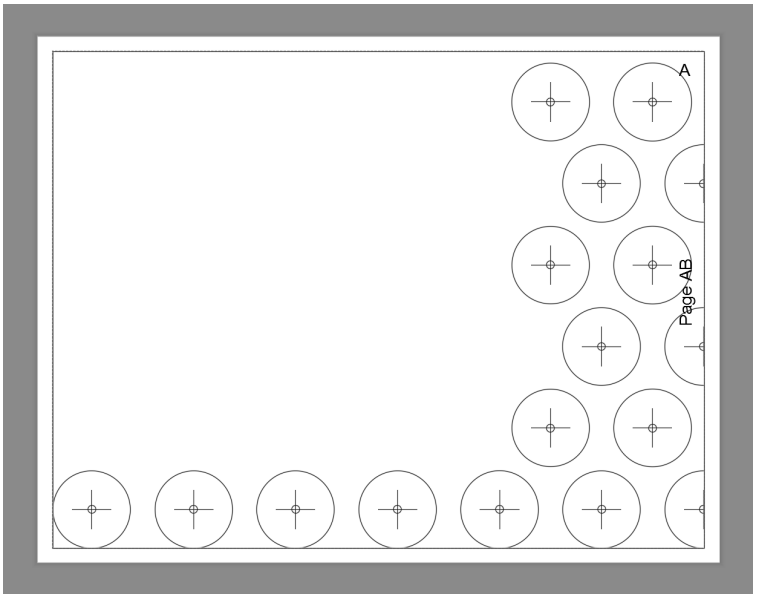

Figure 32: Paper space view of a page from the template for plant flow rate of 20 L/s.

proper alignment between the center of the cluster and base for the best aesthetic. Figure 30 shows how the the template is properly aligned for when the number of orifices in the base is even. Figure 31 shows the alignment for templates with odd number of orifices in the base.

The next step towards finishing the LFOM Template code was to implement it onto the beta server. To put the LFOM Template code onto the beta server, the old LFOM file was replaced in the ADT Designs folder with the new file. Within the new file, both AC.LFOM and AC.LfomOrificesTemplate were expressed as outputs. To make sure the two AC variables produced something on the beta server, an extra line in the Beta Methods file was added under file name and AC variables. The extra line contained the file name LFOMTemp and the corresponding AC variables for the template. Committing to the beta server caused the template to upload as a .dwg file.

Committing the file to the beta server also revealed several critical things. The first thing that was noticed was for certain flow rates, the orifice cluster was not properly aligned with the bottom row. This led to a reevaluation to the origin definition, which revealed that the number of orifices in the row immediately above the bottom row was important for correct alignment. In other words, whether the second row contained an odd number of orifices or even number of orifices mattered. This resulted in testing the eight different combinations between even and odd number of orifices in the base, second, and row with the second largest number of orifices.

While testing the eight different cases and producing the different templates, it was found that all of the templates where the base row had an even number of orifices did not need the code modified from Figure 28. In the case where the base had an odd number of orifices and the row with the next largest number of

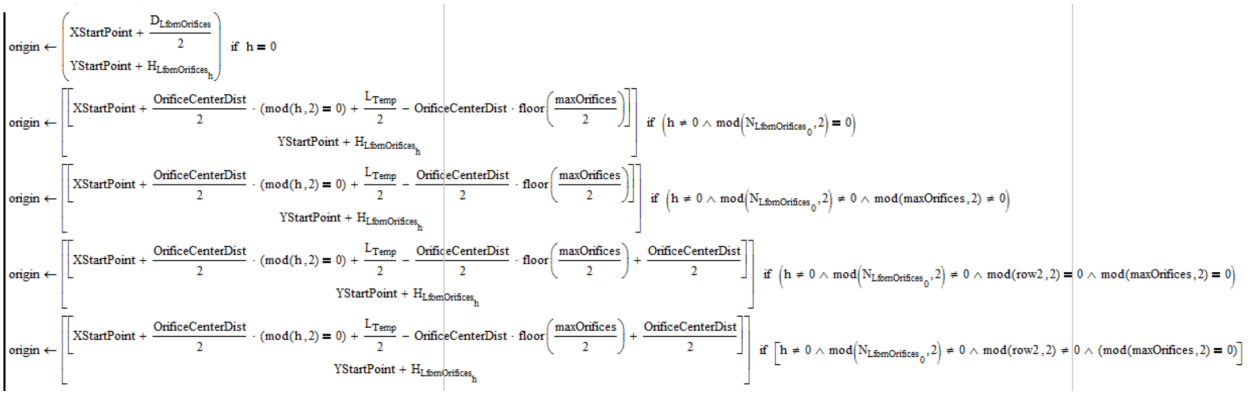

Figure 33: Origin modifications as a result of beta server testing.

orifices had an odd number of orifices, the code also did not differ from that in the third origin definition in Figure 28. The cases that needed modification were the ones where the base had an odd number of orifices and the row with the next largest number of orifices had an even number of orifices.

If the template had a base with an odd number of orifices, a second row with an even number of orifices, and its next largest row with an even number of orifices, the origin definition was the same as the third origin definition in Figure 28 except half the value of OrificeCenterDist was added on to the end. In the other case where the base and second row had an odd number of orifices and the next largest row had an even number of orifices, the origin definition was the same as the second origin definition in Figure 26, but with half the value of OrificeCenterDist added at the end. Figure 33 shows the modified origin definitions

The testing on the beta server combined with the orifice origin modifications also effectively fixed the error with the drawing at 10 L/s.

The culmination of all of these modifications and additions led to the creation of a proper LFOM Template. An example of a final template in model space can be seen in Figure 30 for an LFOM designed for a plant flow rate of 20 L/s. The correctly scaled paper space is exemplified by Figure 32, which shows Page AB of the 20 L/s LFOM Template from the model space.

3 Future Work

There are quite a few, albeit not terribly difficult, tasks that still need to be taken on by the modular team. Regarding the stacked rapid sand filter, the code is fully functional on the beta server indicating that it is ready for partners to use. The only thing left to do for this is to work with Monroe to implement the code on the actual AguaClara server. As a part of this process, the team will have to look into the flow rates where the number of OStaRS/EStaRS filters are changed. Once these key flow rates are identified, the team will upload the modular code onto the actual server using these key flow rates as they show distinct changes in the design. Furthermore, documentation will needed to be checked to make sure everything is up-to-date with the new revisions.

Likewise, the CDCChemTanks modular code has been completed, and can be implemented onto the server that clients can access. Before adding it to the server, however, documentation needs to be written since the CDCChemTanks was a brand new design added to the Final Designs folder. This means there was no prior existing folder that had old documentation that could simply be updated.

Regarding the LFOM and its template, there are a few minor changes that still need to be made. The first task that must be taken on is translating the new template format onto the 3-D model of the LFOM. One of the changes made by the team was to shift the stack of orifices on the template so that they were centered on the page. This change needs to be reflected on the 3-D model as the current model has all of the orifices shifted to the left. Additionally, the team still has to figure out the best way to have partners print out the template of actual use. The eventual goal is to get the server to return a pdf file of the template so that partners can merely open the pdf and print it without any worries. Unfortunately, this is harder than it sounds as the printing feature of AutoCAD is rather complicated. In the meantime, the team wants to put

together detailed, easy to understand instructions outlining the exact steps that partners need to take after opening the AutoCAD file of the template to print the design out. These instructions will likely be posted in a word document that would also be committed to the server and returned when the user requests the template design.

Another challenge for future semesters would be to check the status of all the other modular designs that were not addressed this semester. These would include the flocculator and sedimentation tank designs. The objective should be to inspect the code and check that they function properly by first implementing them onto the beta server. After approving the functionality of the designs, the documentation should be updated to reflect the new pertinent information for the clients. Once all of the designs have been modified and documentation updated, the team should consult with Monroe to get verification to upload the designs onto the server that clients can access.

4 Task Map

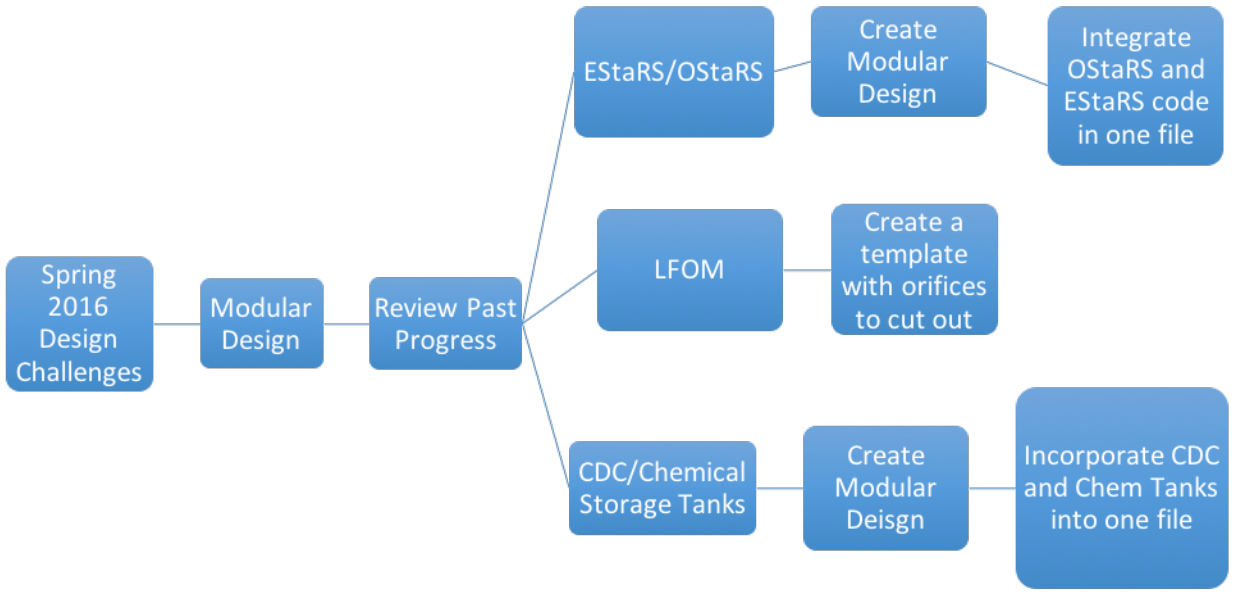

Figure 34: Task Map with challenges

4.a Task Map Details

- Modular Design
  - Review progress made in the fall and determine what still needs to be accomplished
    - LFOM – Create a template for the modular design showing the drilling pattern for the LFOM
    - CDC – Create a stand-alone module with Chemical Storage Tanks integrated
    - EStaRS/OStaRS – Create a module that will draw either EStaRS or OStaRS given specific values; Should try to integrate it into one MathCAD code possibly with an if-statement