Autonomous Cryogenic Load Operations: Knowledge-Based Autonomous Test Engineer

J Nicolas Schrading

Kennedy Space Center

Major: Computer Engineering

KSC FO Summer

Date: 5 8 2013

Autonomous Cryogenic Load Operations: Knowledge-Based Autonomous Test Engineer

J Nicolas Schrading[1]
Rochester Institute of Technology 1 Lomb Memorial Drive, Rochester, New York, 14623

The Knowledge-Based Autonomous Test Engineer (KATE) program has a long history at KSC. Now a part of the Autonomous Cryogenic Load Operations (ACLO) mission, this software system has been sporadically developed over the past 20 years. Originally designed to provide health and status monitoring for a simple water-based fluid system, it was proven to be a capable autonomous test engineer for determining sources of failure in the system. As part of a new goal to provide this same anomaly-detection capability for a complicated cryogenic fluid system, software engineers, physicists, interns and KATE experts are working to upgrade the software capabilities and graphical user interface. Much progress was made during this effort to improve KATE. A display of the entire cryogenic system's graph, with nodes for components and edges for their connections, was added to the KATE software. A searching functionality was added to the new graph display, so that users could easily center their screen on specific components. The GUI was also modified so that it displayed information relevant to the new project goals. In addition, work began on adding new pneumatic and electronic subsystems into the KATE knowledgebase, so that it could provide health and status monitoring for those systems. Finally, many fixes for bugs, memory leaks, and memory errors were implemented and the system was moved into a state in which it could be presented to stakeholders. Overall, the KATE system was improved and necessary additional features were added so that a presentation of the program and its functionality in the next few months would be a success.

I. Introduction

KATE, the Knowledge-based Autonomous Test Engineer, initially began with lofty goals of becoming an advanced Artificial Intelligence (AI) for anomaly detection in any hierarchal system of inputs and outputs. To further these goals, the original architects of KATE used the programming language LISP. Excellent progress was made, with the system capable of making decisions about what component or components could have possibly failed in order to cause the symptoms that it was seeing in a simple water-based fluid system. Although it wasn't an AI as one would think of today, it was impressive to see its reasoning abilities in action.

At this point, the engineers decided it would be best to port the KATE system from a LISP code base running on a proprietary LISP machine to a more open C++ code base running on any GNU/Linux machine. They began this port and successfully converted most of the functionality, with a few extras from the LISP code missing. Unfortunately, the program was cancelled and the work was halted in the early 90s. KATE remained stagnant until the mid-2000s when newfound interest in its capabilities began. In order to demonstrate its powerful reasoning and anomaly detection skills, it was decided that KATE should be converted to analyzing a complex cryogenic system for loading from storage to vehicle tanks. If KATE can provide robust and accurate anomaly detection for this extremely complex system, it will be proven to be a powerful and generic system for anomaly detection in any system – capable of being used for anomaly detection all over KSC and other NASA facilities.

As part of the revitalization of the KATE program, more people have been assigned to work on it, including two interns for the 2013 summer term. A vast array of features, bug fixes, and improvements are required for the system in order to be successfully demonstrated, the first of which is in September of 2013. This paper will go over the additions, fixes, and improvements that an intern at the KSC provided to the KATE system over a 3-month tenure.

II. Additional Features Added

The KATE system in LISP was able to go through its knowledgebase of connected components (valves, pipes, sensors, tanks) and automatically create a schematic of those connections. This feature was forgotten about when porting from LISP to C++. The first feature added for the new KATE that was worked on during this internship was

[1] Software Programmer Intern in Support of the ACLO Program, NE-E7, Kennedy Space Center, Rochester Institute of Technology

a program to bring this functionality back. This was done by utilizing the powerful graphing abilities of the program Graphviz[1], the scripting abilities of Python[2], and the already-implemented capabilities in KATE for displaying schematics in the .png format. Two Python scripts were implemented to help with this feature: graphGen.py and xyGen.py. GraphGen.py went through the flatfile, a file containing the information about the components and their connections, and converted it into the .gv format for Graphviz. XyGen.py went through the .gv file, after Graphviz converted it to contain x and y coordinates for each node in the graph, and wrote a .xy file so that KATE could draw update fields on the nodes of the graph generated by Graphviz. Update fields are simply labeled buttons for each component in the system, with values that update over time as the simulation runs. To the right and below are examples of the scripts run on the command line, as well as a screenshot of KATE with the feature fully implemented. Figure 3 shows the .png generated by Graphviz working in conjunction with graphGen.py to get data about the nodes and edges of the directed graph of a connected system. On top of the .png are update fields, placed appropriately by xyGen.py working in conjunction with Graphviz. The manual commands on the command line are unnecessary for users of the KATE system because as part of implementing this feature, automatic system generation was added. This calls all the python and Graphviz commands automatically and displays the generated graph and update fields all at once so that the user does not need to know how to work with these scripts. If, however, the user wishes to view the connected system in a different program, for example YEd, they can run the commands on the command line and use YEd to display the graph, as shown in Figure 1. The next addition to KATE was a search functionality for the schematic viewer feature. This allows the user to look for update fields displayed in schematic viewer, and center the screen on the update field if found. Figure 4 shows an example of this feature in action.

```
[jschradi@katehost2 kate]$ python graphGen.py -h
graphGen.py -i <flatfile> [-o <outfile>] [-f <format>] [--v=boolean] [--l=boolean]
[jschradi@katehost2 kate]$
[jschradi@katehost2 kate]$ python graphGen.py -i ~/dev/kate/sun/exe/spl.11.flatfile -o spl.11.graphml
[jschradi@katehost2 kate]$ YEdExec spl.11.graphml &
[1] 19566
```

Figure 1. GraphGen.py use on the command line.

```
[jschradi@katehost2 kate]$ python xyGen.py -h
xyGen.py -i <dotfile> [-o <outfile>] [--l=boolean]
[jschradi@katehost2 kate]$ dot flatfile.gv -o dot.gv
[jschradi@katehost2 kate]$ neato dot.gv -n2 -Tpng -o spl.11.png
[jschradi@katehost2 kate]$ python xyGen.py -i dot.gv -o spl.11.xy --l=false
```

Figure 2. XyGen.py use on the command line.

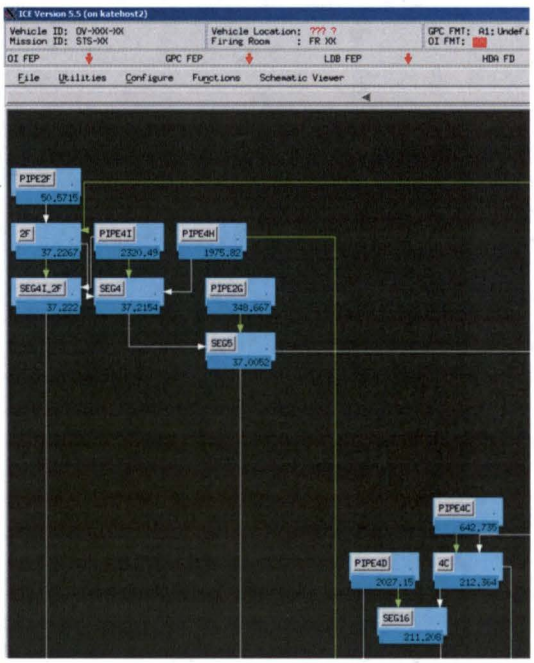

Figure 3. A generated .png from graphGen.py and Graphviz, with update fields placed from xyGen.py within KATE.

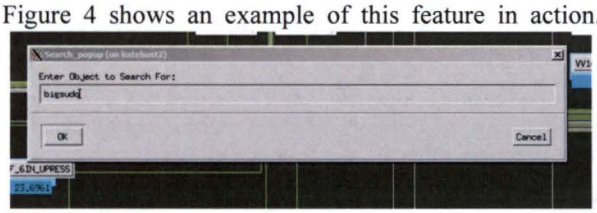

Figure 4. Searching for an object in schematic viewer.

Figure 5 shows the results if the object is found and figure 6 on the next page shows the results if it could not be found. Originally, this search would only work

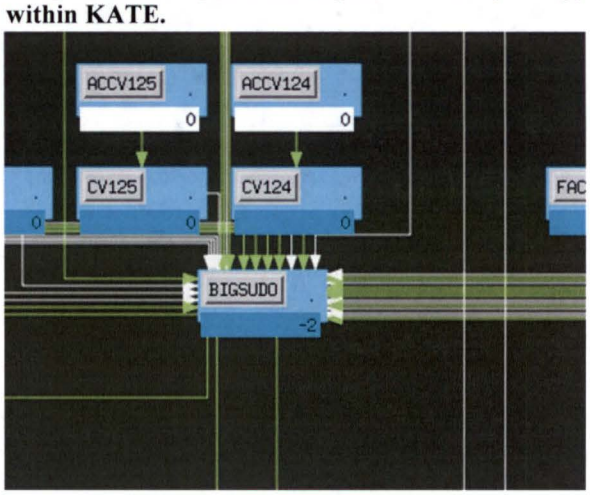

Figure 5. Object is found and screen is centered on it.

if the user entered the exact name desired. A feature added later allowed the user to type in any substring of an update field name and get results in a list. The user can then select from the list the desired object so that the screen centers on that object. Furthermore, all searching is kept in a history so that if the user wants to view and center on previous searches, it can be done quickly and easily. The forward and back arrows in Figure 7 are the buttons that allow the user to move forward and back in search history.

Figure 6. Object could not be found.

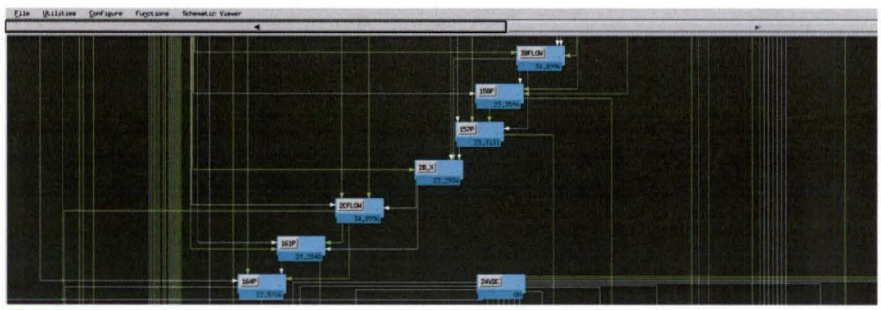

Figure 7. Search history arrows can be seen at the top of the schematic

Another feature added to KATE, and specifically to help the developers, was a python script called killAll.py. Because KATE uses multiple processes for various functionality, e.g., the reasoner for simulating the system and determining possible failed components, the data provider for running through recorded simulations, and the ui for displaying information to the user in a graphical user interface (GUI), when a failure occurs and the program crashes, separate processes may continue running. This caused delays in debugging because the user had to manually search through the list of running processes and kill off any that were continuing their execution. KillAll.py is a simple one line command to kill every process that could be running in the background. It saves time for the developers and improves their workflow.

```
[jschradi@katehost2 exe]$ python killAll.py
killed 2962: ui
killed 2963: ui
killed 2964: ui
killed 2966: reasoner
killed 2968: pdp
```

Figure 8. KillAll.py in action, killing 3 UIs, a reasoner, and a data provider running in the background.

As part of the schematic viewer changes, after the features involving graphGen.py and searching were completed, it was desired to have the ability to display schematics with update fields on them and swap between these schematics easily. To do so, the ability to right click on an update field and select an option called 'View Schematic' was added. This opens up a schematic with that update field on it, if one exists.

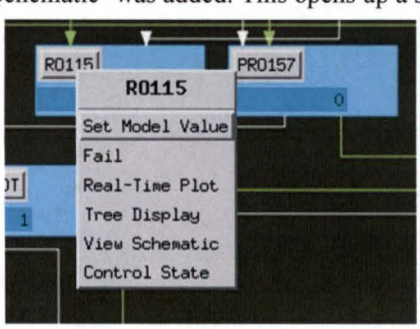

Figure 9. View Schematic added to the right click menu.

Figure 10. Selecting the schematic desired.

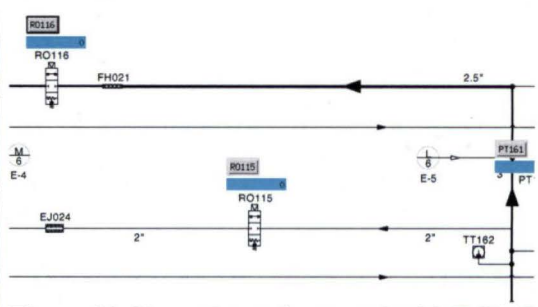

Figure 11. New schematic opened with RO115 in center.

If multiple schematics exist with that update field, the user is prompted for the correct schematic, and it is then opened.

The final major feature added to KATE was the ability to display cryogenic liquid rising and falling within a storage tank, corresponding to the amount of gallons currently in the tank as determined by the reasoner. This feature required quite a bit of research into volumes of ellipsoidal storage tanks, the equations necessary for calculating their volumes and the height of the liquid, and drawing in X-Windows/Motif. A paper[3] by Dr. Dan Jones, a senior process chemist for Stockhausen Louisiana, LLC, was utilized for the equations necessary to calculate the liquid level height. In figure 10, the equations are seen. V_f is the volume of the liquid, D is the diameter of the tank, h is the height of the liquid, and a is the height of the ellipsoidal bottom. All units are in inches (V_f is in cubic inches). An iterative algorithm was written to calculate the volume in the tank, given a test liquid level height and the dimensions of the tank. If the volume was too small compared to the actual volume in the tank, the test height was incremented and the algorithm repeats. This successfully determines the correct height within the tank. From this information, the tank was drawn in KATE using the built-in X-Windows/Motif drawing tools like XfillArc() and XfillRect().

Ellipsoidal bottom.

$$V_f = \begin{cases} \frac{\pi}{4}\left(\frac{Dh}{a}\right)^2\left(a-\frac{h}{3}\right) & \text{.....} \quad h < a \\ \frac{\pi D^2}{4}\left(h-\frac{a}{3}\right) & \text{.....} \quad h \geq a \end{cases}$$

Figure 10. Equations necessary for calculating liquid level height.

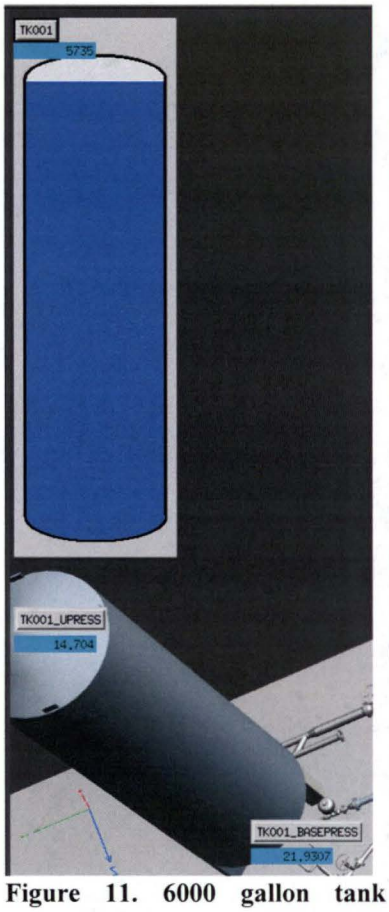

Figure 11. 6000 gallon tank almost full.

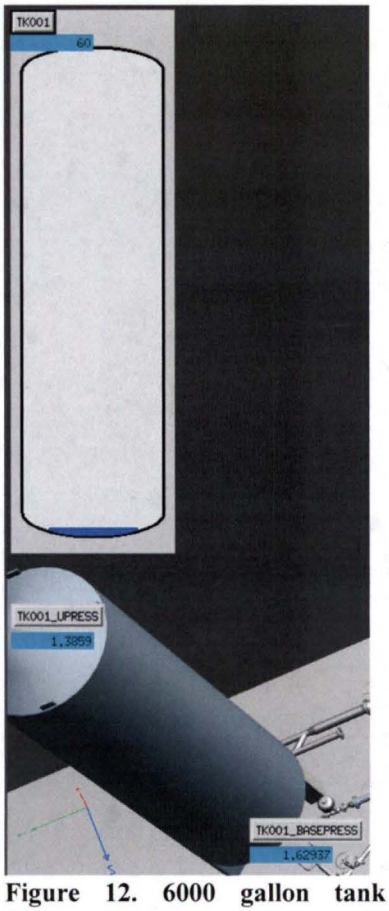

Figure 12. 6000 gallon tank almost empty.

III. Improvements and Bug Fixes

During the implementation of the Graphviz overall schematic viewer feature, it was noted that the schematic viewer tool, already implemented by a previous KATE engineer, had many critical bugs and faults. It would crash if the user attempted to: open multiple schematics for viewing, switch between schematics with update fields on them, remove schematics with update fields on them, load a schematic with update fields when the system doesn't have a knowledgebase file loaded, and display update fields off the screen. All of these faults were found and fixed so that the user would have a much more stable and professional experience.

After this fix was completed, it was desired to have the ability to left click on update fields and view their first level inputs and outputs, as well as some additional information about their state. This feature was already in KATE; however, it was unstable and did not provide a clean, easy to read information pane. This code was overhauled and the pane was reworked to read much more easily and professionally. Figure 13 on the next page shows a before shot, and Figure 14 on the next page shows an after shot. During investigations into why this specific feature was so unstable, causing crashes that were very hard to trace, an important discovery was made. The linked list implementation that was being used had a bug in its logic for adding and removing nodes. When a node was added or removed, its previous node was not being appropriately set, leaving an undefined (but sometimes valid) space in memory as its previous node. Since the previous slot was not used very often, this bug went unnoticed since the early 90s. It was causing segmentation faults in the left click info pane, though. This bug was tracked down and fixed, and the left click info pane became stable. This feature was also inappropriately causing memory leaks when the user closed the pane, because the update fields were not being deleted. This was noted and fixed as well.

Another problem noted during the internship was the prevalence of memory leaks, invalid memory accesses, and various other memory issues stemming from the inappropriate use of pointers. Many of these problems were noted

and fixed when they were found. Examples include changing the delete keyword to delete[] when deleting arrays of pointers, and making sure to use character pointers carefully, or simply switching from character pointers to strings, because they avoid the hassle of managing memory manually. A huge problem in the codebase was the use of strstreams to buffer in character pointers. Strstreams are old and deprecated character pointer stream buffers that caused memory leaks if the programmer was not very careful about their use (which was difficult and confusing, so oftentimes memory leaks were unavoidable). An effort was made to remove all strstreams and replace them with stringstreams. Every occurrence of strstream in the ui module was replaced, and most other occurrences were replaced as well, although at the time of writing this report, some are still in the codebase. After fixing many memory leaks and errors, a test was run in the memory analyzing program valgrind with the old codebase versus the new codebase. Figures 15 and 16 compare the results. As can be seen, significant improvements in the amount of memory errors reported were made, and a small improvement in the number of definite memory leaks was made.

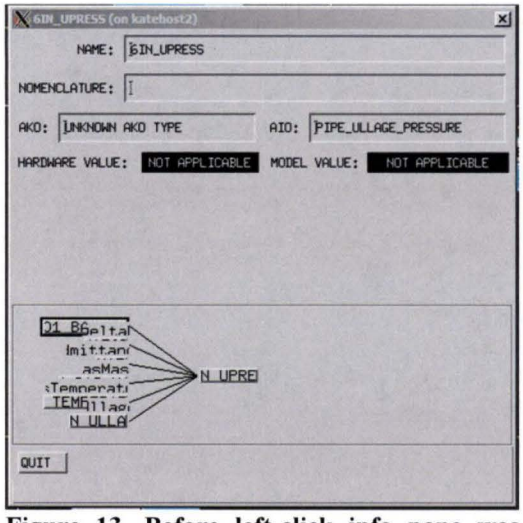

Figure 13. Before left-click info pane was reworked.

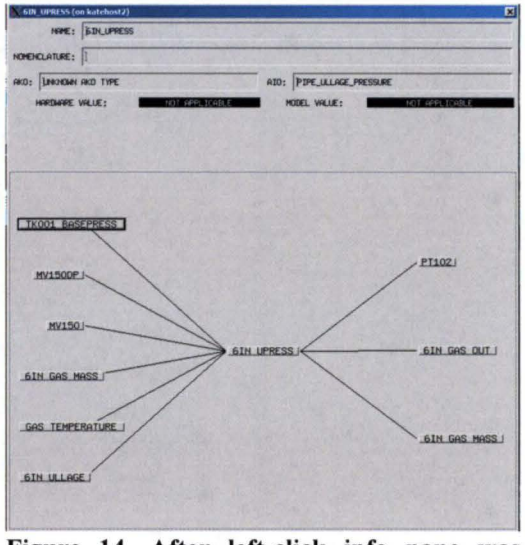

Figure 14. After left-click info pane was reworked.

```
==10078==
==10078== HEAP SUMMARY:
==10078==     in use at exit: 1,910,267 bytes in 21,816 blocks
==10078==   total heap usage: 78,108 allocs, 56,292 frees, 208,943,999 bytes allocated
==10078==
==10078== LEAK SUMMARY:
==10078==    definitely lost: 12,387 bytes in 621 blocks
==10078==    indirectly lost: 6,118 bytes in 514 blocks
==10078==      possibly lost: 148,154 bytes in 5,998 blocks
==10078==    still reachable: 1,743,608 bytes in 14,683 blocks
==10078==         suppressed: 0 bytes in 0 blocks
==10078== Rerun with --leak-check=full to see details of leaked memory
==10078==
==10078== For counts of detected and suppressed errors, rerun with: -v
==10078== Use --track-origins=yes to see where uninitialised values come from
==10078== ERROR SUMMARY: 535 errors from 5 contexts (suppressed: 30 from 7)
```

Figure 15. Before memory changes. 535 errors detected, 12.4KB memory definitely leaked.

```
==9684==
==9684== HEAP SUMMARY:
==9684==     in use at exit: 1,836,892 bytes in 20,074 blocks
==9684==   total heap usage: 73,820 allocs, 53,746 frees, 207,671,121 bytes allocated
==9684==
==9684== LEAK SUMMARY:
==9684==    definitely lost: 11,782 bytes in 1,065 blocks
==9684==    indirectly lost: 6,118 bytes in 514 blocks
==9684==      possibly lost: 174,747 bytes in 6,917 blocks
==9684==    still reachable: 1,644,245 bytes in 11,578 blocks
==9684==         suppressed: 0 bytes in 0 blocks
==9684== Rerun with --leak-check=full to see details of leaked memory
==9684==
==9684== For counts of detected and suppressed errors, rerun with: -v
==9684== ERROR SUMMARY: 4 errors from 1 contexts (suppressed: 29 from 7)
```

Figure 16. After memory changes. 4 errors detected, 11.8KB memory definitely leaked.

An additional improvement over the old KATE program was the UI. The old UI had references to many shuttle-era and older pieces of information and were therefore unnecessary. These references were removed and replaced with their 2013 counterparts, helping to modernize the user-facing features of the program.

Improvements were also made to the tree display feature in KATE. This is a larger version of the left click info feature with multiple levels of inputs and outputs being displayed. This feature had many crashes, especially when doing something slightly outside of the normal flow of execution. These crashes were found and fixed, and it is now stable and robust as well.

One final, large improvement of KATE involved the removal of copious amounts of compiler warnings. Almost every module had at least one compiler warning, usually relating to the deprecated conversion from constant character pointers (string literals) to character pointers. With the help of the other intern, every compiler warning was abolished, except for in alo-midlevel.C, a class that was in heavy use by the other developers. At first, casting was used to fix the deprecated conversion warnings, but then it was decided that it would be cleaner and more efficient to change the functions with warnings to take constant character pointers, since that is really what they should have been in the first place. This was tedious work and required changes all over the codebase, but it significantly improved the compilation and safety of the code.

An example of the compiler warnings can be seen in figure 17.

Figure 17. Compiler Warnings.

IV. Conclusion

When work began on KATE at the start of the Summer, it had memory errors and memory leaks, segmentation faults in many parts of the code, compiler warnings, and outdated and hard-to-read UI components. Now it is much more stable, robust, warning-free, and has fewer memory errors and leaks. It has new, helpful features that the engineers and developers use daily to visualize their work and improve their workflow. Its stability in features that used to crash often is saving time and improving its user friendliness and image for observers. All of these additions and improvements will help in September when KATE is reviewed and tested. Although there is still much work to be done, these 3 months have had a significant impact on KATE.

Acknowledgments

Nicolas Schrading thanks Charlie Goodrich and Bob Panzak for helping him become accustomed and familiar with KATE and its goals, as well as NASA in general. He also thanks Charlie and Bob for being especially enthusiastic about his work.

He thanks Ed New for going over the history of KATE and CUIs, as well as talking about cool space and technology things.

He thanks Barbara Brown for extending his amazing experience at NASA (and playing a part in hiring him in the first place), and Felix Soto Toro for being his mentor and offering him the position.

He thanks Mitch Bolling for providing his expert guidance on makefile creation and debugging, as well as general C++ practices.

He thanks Walter Wehner for being an excellent co-worker, providing an extra set of eyes on difficult problems, and showing him the powers of C#.

He thanks all the workers in the KSC Education Office for providing him with the opportunity to work at KSC.

Finally, he thanks all the wonderful people he has met during his time at NASA, especially if they showed him cool things like robot arms or rockets.

References

[1]Graphviz, Software Package, Ver. 2.30.0, AT&T Labs Research and Contributors, Austin, TX, 1988.

[2]Python Programming Language, Software Package, Ver. 2.6.6, Python Software Foundation, DE, 1991.

[3]Jones, Dan, "Calculating Tank Volume: Saving Time, Increasing Accuracy" *Webcalc*[online database], URL: http://www.webcalc.com.br/blog/Tank_Volume.PDF [cited 5 August 2013].