Software Development Cycle and Version Control Improvement

(NASA JPL SUMMER 2019 INTERNSHIP PROJECT)

Arash Hosseini Jafari

Mentors: Abhinandan Jain & Jonathan Cameron

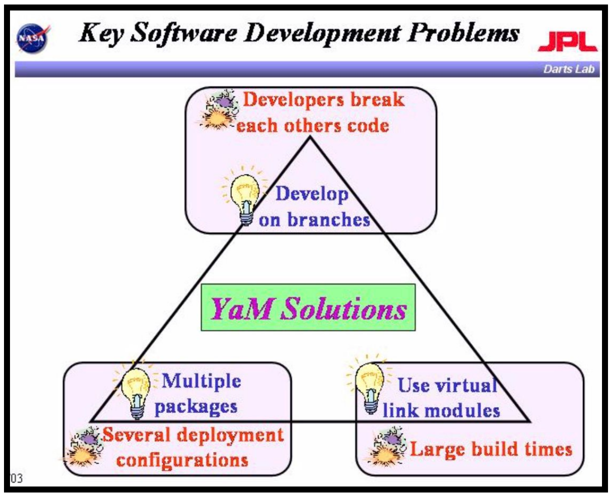
Abstract:

The Mobility and Robotic Systems Section's Dynamics And Real-Time Simulation (DARTS) lab develops advanced high-fidelity, multi-mission simulation tools for the closed-loop development and testing of spaceflight systems. The DARTS Lab provides multimission support due to its highly modular simulation framework which is used by many missions at JPL and elsewhere at NASA. This modularity is facilitated by an inhouse con’

current software development tool called Pyam. My work on the Pyam codebase during the summer involved implementation of various improvements and features to allow increased productivity throughout the software development cycle. These features included the development of regression tests, implementation of automated email alerts related to version control failures, improvement of the management of software modules by resolving issues with their addition and removal to the development environment (the sandbox), and the addition of an intricate maintenance feature to facilitate the delivery of delta updates and patches to NASA project teams that must remain on older, or mission specific, versions of the lab’s simulation tools.

Introduction:

The Dynamics And Real-Time Simulation lab develops advanced high-fidelity, multi-mission simulation tools for the closed-loop development and testing of spaceflight systems. (1) The DARTS Lab provides multimission support due to the highly modular simulation framework. The concepts of modularity, configurability, and reusability are also implemented in development of the simulation software itself through YaM and Pyam. (2)



YaM fosters a workflow that supports reusable software development infrastructure. Yam branches make concurrent software development, with multiple adaptations of overlapping code possible. It prevents developers from stepping on each other's work while simultaneously working on a codebase. With branched software development and the notion of packages it allows deployment of different configurations. Moreover, the use of virtual link modules & sandboxes allows long build times. With a standardized make & module level makefiles, (2) YaM also allows the seamless integration of multiple third party software dependencies, targets, platforms, sites, and operating systems. Ultimately YaM allows developers to “release early, release often” while streamlining incremental software testing and development. (3)

Methods and Implementation

At the outset of my internship I began familiarizing myself the code base as well as the file structure of the Pyam software. Given the large and complex nature of the Pyam it was much easier to familiarize myself with certain aspects as I solved existing problems and added each specific feature. The first task that I worked on first was creating a test case for a specific functionality called no-keep-release. No keep release is a command that inhibits the save function of Pyam from moving modules the release area. It is a future that is used by developers that are working remotely and on laptops. After exploring the regtest scripts I discovered that they were created using a third-party tool called Cram. Reading and understanding the Cram documentation allowed me to not only complete the above-mentioned ask but facilitated the development of many of the future tasks and I will discuss here. It allowed me to quickly and easily test my code, refine it, test again, and repeat until the desired result was reached.

The next feature that I worked on was the addition of email alerts that would be triggered when there is a failure of the build function in Pyam.In order to accomplish this task there were several concepts that I needed to familiarize myself with. Working on creating the email alerts I learned about SMTP servers, ports, and sockets, as well as some of the packages in Python that allow scripts to initiate an email message. While such a task may seem trivial on the surface, it proved to be rather time-consuming. I wanted to implement the email alerts in a way that was consistent with how emails were initiated by the program in other circumstances, such as the release emails. The program has several interconnected scripts that handle the emailing capability as well as generating the proper credentials and configuration for the mail server which made the task challenging.

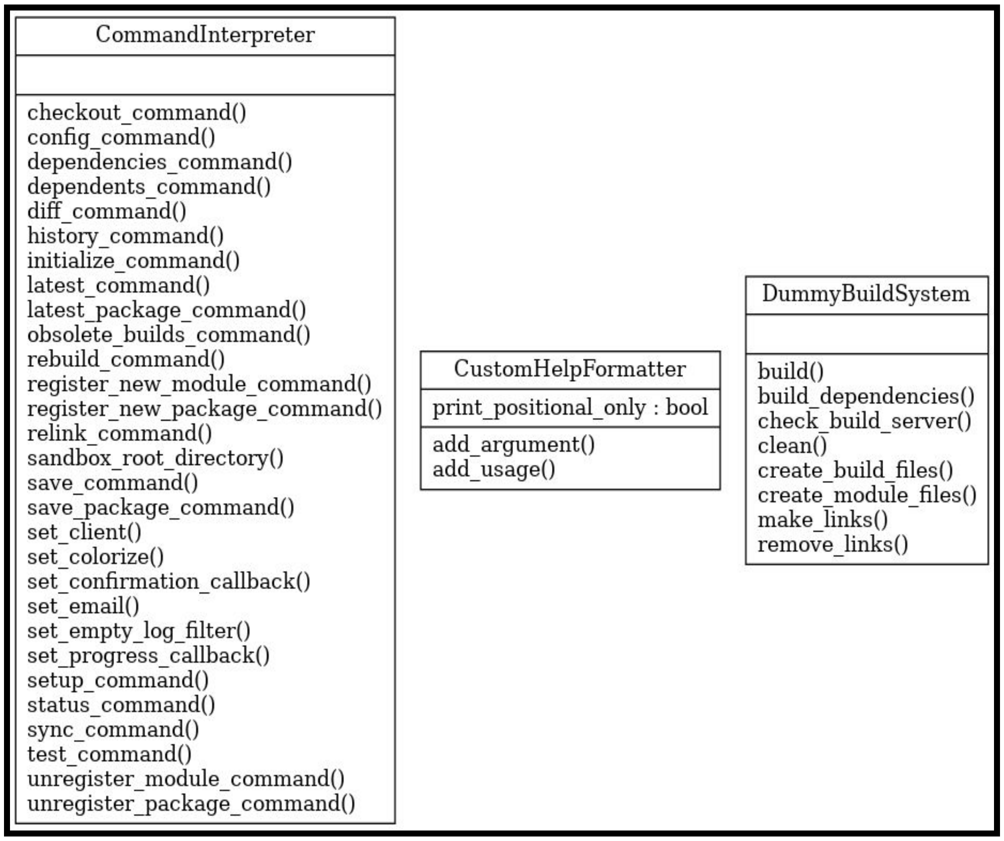


Figure 1: The classes within Pyam script and their respective function.

The next issue I was tasked with fixing was regarding the management of modules within the sandbox. In particular, some users experienced issues deleting modules from their sandboxes and then checking out new/fresh copies of the module back into their sandbox. The issues were most pronounced when specific branches of the module was checked out using the pyam --checkout branch command. More specifically the issue was that the configuration file of the software was not updated to reflect the fact that the workmodule was removed from the sandbox. The command pyam checkout ---link was thought to contain functionality to alleviate this issue by making the proper changes to the configuration file. However this checkout --link command was not working in this way.

After working on replicating the issue for several days while also exploring all the functions, classes, and packages that facilitate checkout, I discovered a little known function in the codebase that solves the majority of the issues. The config_to_link function in the client.py script was able to reconfigure the configuration file automatically to allow the removal and re-checkout of a module. I also discovered config_to_work command which could be used in the reverse scenario. After this discovery, I added helpful error messages that explained the proper commands to run to reconfigure the sandbox config file and fix any discrepancies. I will also be adding a dedicated module scraping function in the next update to further automate this process.

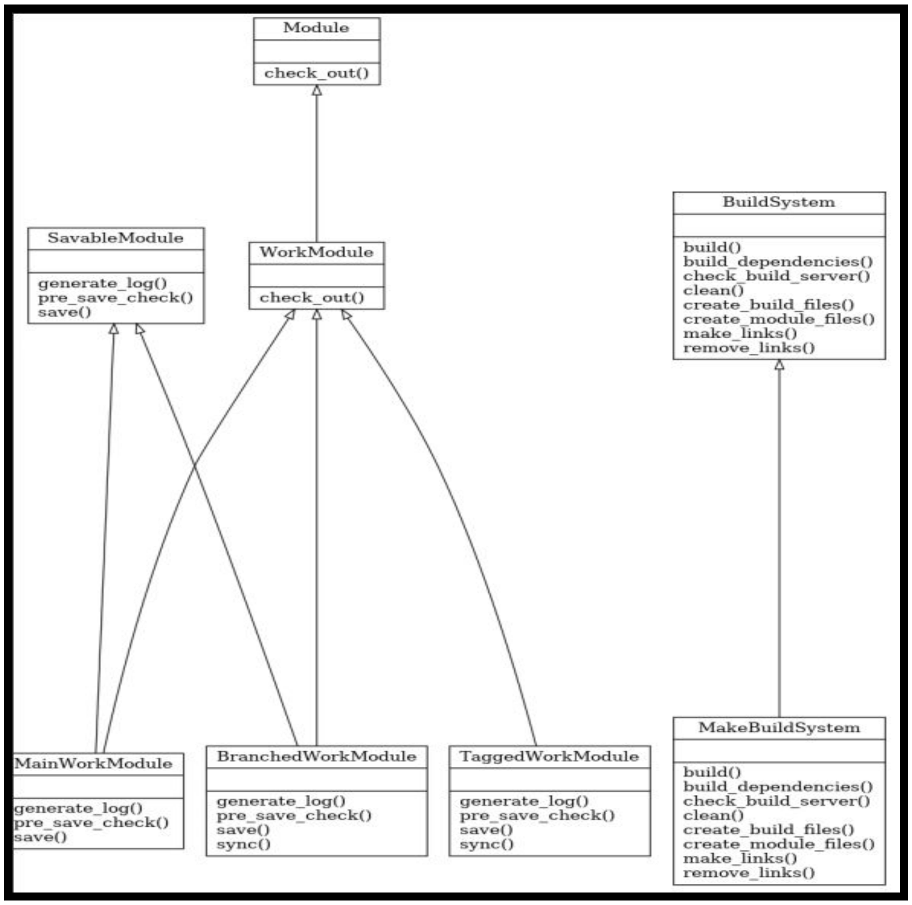


Figure 2: Pyam packages and their functions pertaining to the save functionality

The final and most complex task I undertook was the addition of a feature to Pyam to allow for maintenance branches to service specialized branches of the DARTS software. This feature used to exist in the previous versions of the software, however it is absent from the recent pythonic version. The maintenance feature is useful for creating and maintaining project-specific/task specific versions of the software that DART's produces. Many projects and groups that utilize DARTs software do not neccessarily want the latest versions of our software, which may include un-vetted, undocumented, experimental features that may distrupt their current workflow.

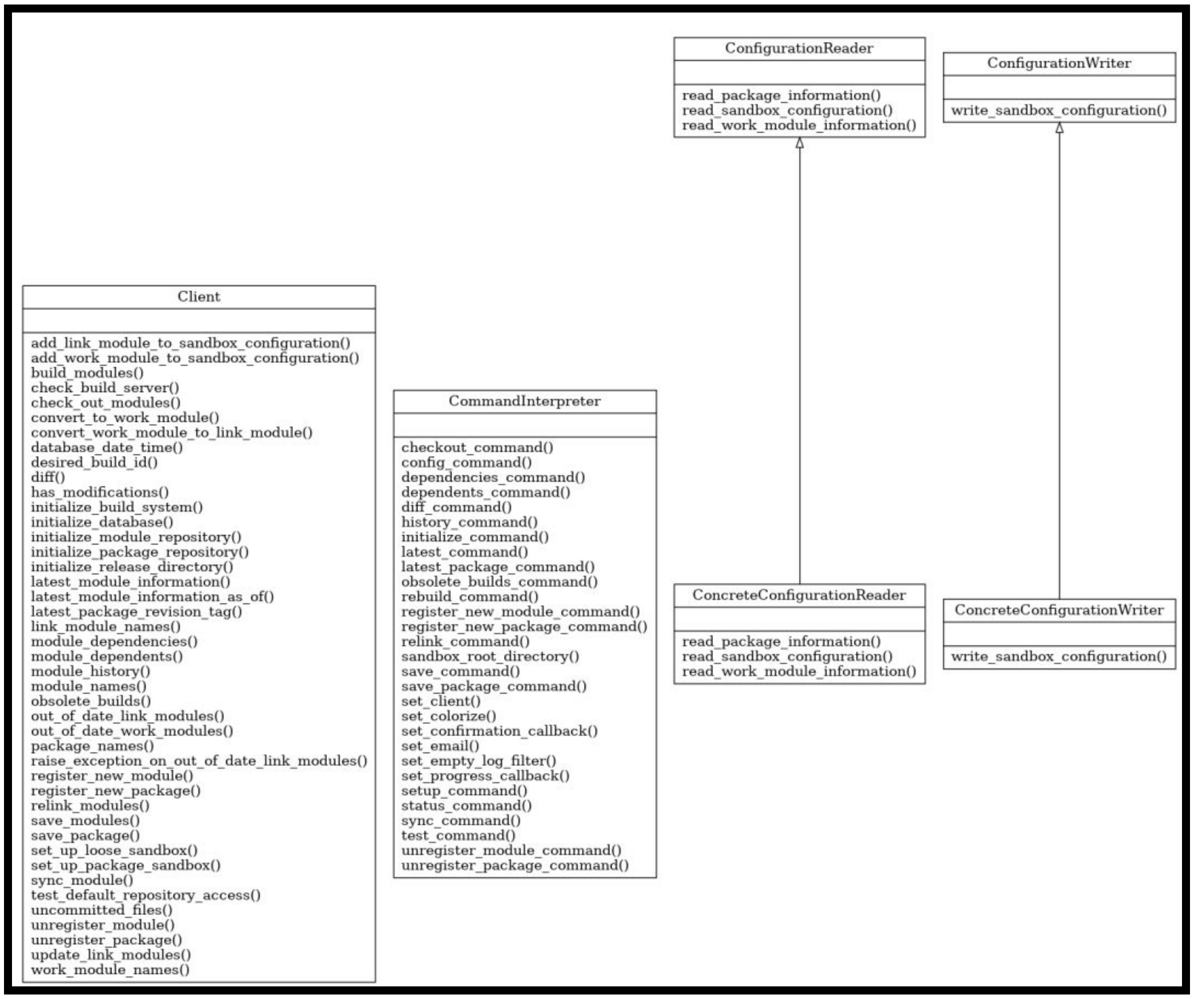


Figure 3: Functions within Client.py and scripts regarding the alteration of pyam configuration.

The maintenance feature would allow the creation of serviceable, project specific, versions of DARTS software that do not merge with our latest branch of the software when pyam save is run. With such a feature, specific point changes and fixes can be implemented in the version of software used by a given project (ex: Mars2020).

Furthermore, these ‘delta-change’ can be done without having to update the software used by the project team to the absolute latest version of the software in our release area.

There has been significant progress made towards the completion of this task thus far. I initially spent a great deal of time tracking down the scripts, classes, and functions necessary to add the maintenance feature. I have since created a robust test-case to facilitate further development. The checkout functionality has also been altered to successfully create a maintenance branch when the --maintenance command is given. In its current state this feature is cable of saving the branch in addition to checkout. To build the Pyam maintenance function I have written scripts that recognize a maintenance package and alter the revision tag so that the module is labeled and saved as a maintenance branch in the release area. The scripts also bypass merging with the main trunk. Furthermore, the code prevents the maintenance package from becoming designated as the latest revision.

Future Work:

Future work should focus on improving the functionality of maintenance branch by reducing the number of manual modifications to the configuration file in Pyam needed to retrieve specific branches and releases of modules.

References:

1) DARTS Lab https://dartslab.jpl.nasa.gov/
2) YaM https://dartslab.jpl.nasa.gov/YaM/index.php
3) A. Jain and J. Biesiadecki, "YaM - a framework for rapid software development," *2nd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT'06)*, Pasadena, CA, 2006, pp. 10 pp.-.doi: 10.1109/SMC-IT.2006.89