26th International Congress on Modelling and Simulation, Adelaide, SA, Australia, 30 Nov–4 Dec 2025
mssanz.org.au/modsim2025

Redesigning the APSIM test and deployment system to bring benefits to developers

Andrew Paroz [a], **Julian Rich** [a], **Dean Holzworth** [a], **Neil Huth** [a]

[a] *CSIRO Agriculture and Food, Forest Hill, Queensland, Australia*

Email: andrew.paroz@csiro.au

Abstract: Modelling software systems like APSIM are required to be both consistent and largely error free, and therefore require extensive testing on any changes before they are added. The existing automated testing suite for APSIM was slow and unsuitable for the modern development process so a replacement system was designed to replace it. The new system produced a significant drop in costs, faster time to run tests, reduction in system complexity and improvements in user experience for collaborating developers. This was made possible by moving to a distributed cloud system for testing, consolidating systems to reduce interactions and centralising outputs to one place.

Keywords: *APSIM, Software process, Automated testing*

1. INTRODUCTION

APSIM (Agricultural Production Systems sIMulator) is a software system that allows complex modelling of agricultural systems (Holzworth et al. 2018). Automated testing is essential to ensure APSIM functions as intended and generates reliable data efficiently—both in its underlying code and its model outputs. As regular contributors, model complexity and number of stored simulations have increased, slowdowns and inefficiencies have been exposed within the existing automated testing and builds systems, necessitating a redesign of the system.

To address these issues, key changes were identified that the new system would need to implement. Required features include moving from a single high-performance virtual machine to a scalable distributed cloud system for simulation testing, allowing for multiple contributors to test new features concurrently, reducing the overall complexity of the system for maintainability and centralising output for contributors to one place to make onboarding easier and less confusing.

2. AUTOMATED ARCHITECTURE

In 2018 the APSIM code repository adopted an off the shelf system known as Jenkins to run a suite of testing and build tasks so the software could move to a system of continuous integration (Holzworth et al. 2018). Jenkins would automatically build and run pull requests using a high performance 96-core virtual machine for simulation tests, and smaller four-core system for unit tests. Results were reported back to an Acceptance Test server for displaying to users, while output logs and errors were visible on the Jenkins webpage. Release builds of APSIM were created whenever a pull request was merged, which was detected by the build's server and would signal Jenkins to create Windows, Linux and MacOS installers, and a Docker image. These installers were then stored in the build server for distribution.

As shown in Figure 1a, this led to quite a complex system with information for users spread across different services, and workflow scripts and configurations across multiple code repositories. This caused lengthy maintenance delays when an error occurred, inflated costs for running a high-performance virtual machine, confusion for users around why their pull request failed and restrictions on the types of tests that could be run, such as emulating the user interface.

To address these issues, a new automated system was designed and developed. GitHub workflows were created to replace the functionality of Jenkins, which allows for unit tests to be run, and release builds created entirely within the GitHub service. Separate GitHub workflows allow for unit testing in both Windows and Linux environments, exposing any operating system specific issues.

Instead of running the entire model test set on one powerful machine, the jobs are broken into small parts and distributed across an Azure compute cloud of low power machines. A pull request specific APSIM docker image is created and used to run the validation set. Afterwards these machines then report back to the Acceptance Test site which now shows both the statistical results of any changes to the models, along with the output logs. For release builds, the installers and docker image are created on GitHub in an action and then pushed to the builds

server and docker repository. As the test data set continues to grow in the future, this design will allow the system to maintain its current speed by increasing the number of virtual machines the work is spread across.

This improved design reduces the complexity of communication between the different components, as seen in Figure 1, replacing most two-way communication between services with singular loops and moving most of the testing system scripts back into the main APSIM repository. When reviewing errors in pull requests, users now only have two places to look, either at the unit test logs run within GitHub, or at the Acceptance Test website where the simulation logs and statistics are visible.

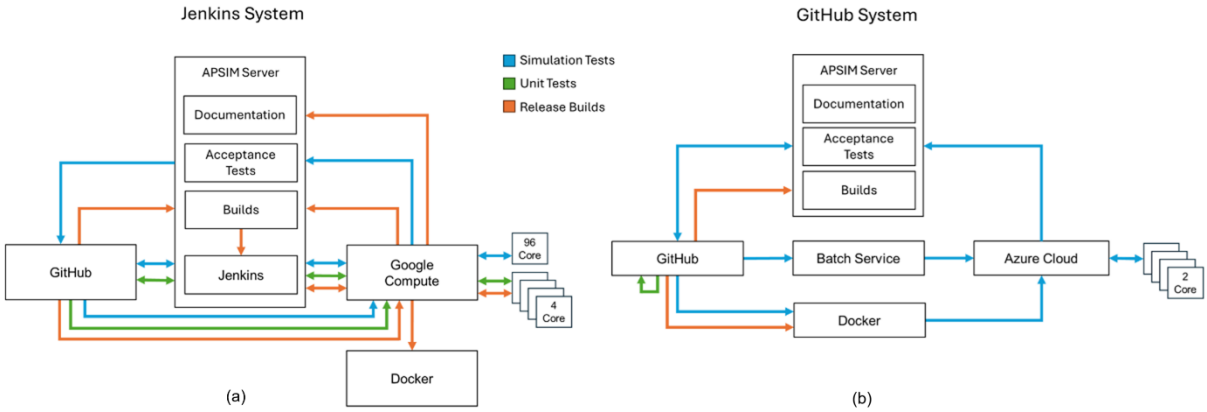

Figure 1: Information flow between components in the Jenkins system (a) and GitHub system (b)

3. RESULTS

After running both systems concurrently to ensure the functionality of the Jenkins system was replicated correctly, the GitHub system showed significant improvements. By removing the expensive 96 core virtual machine, costs dropped from around $1500-1800 a month to just $300-350. Runtimes were also reduced from 30-35 minutes to under 15 minutes, as more simulations could be run concurrently on a distributed cloud.

The information required by our scientific contributors is now centralised on the Acceptance Tests webpage linked directly from GitHub. This system also allows multiple pull requests to be run at the same time, allowing for faster turn arounds when testing model changes, or when adding new features. As many of our science contributors are not software engineers, giving them an easier way to interact with the automated testing allows them to work more efficiently and requires less assistance from the software development team.

The GitHub system has also allowed the configuration and code for the testing system to be brought back into the main repository for APSIM, rather than spread across several repositories. Inter-component communication can now generally be conducted in just one direction, and components like the documentation system can run in an autonomous way without needing to be a part of the testing or builds systems. This has removed duplicated code between repositories, centralised unit testing for the project, reduced the overall complexity of the testing and builds workflows, and generally improved the maintainability of the testing system.

4. CONCLUSIONS

Any long-lasting software with regular updates like APSIM will encounter scope creep and inconsistencies as features are added over time, which is why automated testing systems are so important. Those automated systems, however, also tend to grow and change, and require refactoring just as often as the software they test. The changes we've outlined have allowed us to significantly reduce costs, increase performance and improve the user experience for contributors. These changes are forward-looking, with the ability to scale up the system as more tests are added and model processing increases.

ACKNOWLEDGEMENTS

Acknowledgment is made to the APSIM Initiative who funds the software development of APSIM and guides the direction of this project.

REFERENCES

Holzworth DP, Huth NI, Fainges J, Brown H, Zurcher E, Cichota R, Verrall S, Herrmann NI, Zheng B and Snow V (2018) APSIM Next Generation: Overcoming challenges in modernising a farming systems model. Environmental Modelling & Software, Volume 103, pp 43-51, https://doi.org/10.1016/j.envsoft.2018.02.002.

