SAND2021-14650C

A Testing Strategy that Supports Scientific Software Sustainability

James M. Willenbring - jmwille@sandia.gov
Sandia National Laboratories, North Dakota State University*

1. The Challenge of Testing a Scientific Software Ecosystem

Testing scientific software packages is challenging. Many test suites that were not designed with modern systems in mind, and therefore are fragile. Some packages have system-level tests, but lack unit and integration testing. Several scientific software projects have insufficient testing of any kind. These kinds of problems are not new and are broadly recognized.

More recently, it has become clear that while testing a scientific software package is difficult, testing an ecosystem of scientific software is far more complicated. Achieving a single state where all of the packages in the ecosystem can coexist (for example, two packages do not require different versions of the same dependency) is challenging. Maintaining this state in the face of new development is much harder. There are two fundamental ways that new development can be dealt with.

The first option is to test release versions of each package. The advantages to this approach are that the software is generally not changing quickly, and release versions tend to be fairly stable. One disadvantage of this approach is that changes come in large batches and it can be time consuming to identify which change led to a failure. After a problem is identified, fixes might be complicated because significant development may have occurred since the change that caused the failure. A second disadvantage of this approach is developers or users may want to use a pre-release version of one or more packages.

The second option is to test development versions of each scientific software package. This approach allows for the flexibility of testing changes individually or in small batches, but requires more complicated communication patterns, policies, and workflows to execute effectively. If one development version changes in a way that breaks another, should the test be allowed to fail, or should the version not be updated until all builds and tests pass? What if a test is failing on only one platform? What if a change to one package reveals a bug in another? What if the team behind one package fails to respond to bug reports? These questions are representative of the many decisions that must be made in calibrating the orchestration of testing for an entire scientific software ecosystem.

2. Designing a Testing Strategy

The ultimate solution includes both release and strategic development testing. While creating a complete strategy for testing is complicated, defining the policies, workflows, and funding model

**Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the US Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. SAND2021-XXXX C* to support the strategy is even more challenging. However, until the testing is in place, it is hard to evaluate policies and workflows. Over the past few years, a lot of progress has been made towards improving the stability and sustainability of the scientific software ecosystem via better testing. At the ecosystem level, the Extreme-Scale Scientific Software Stack (E4S) is leading the way with a growing validation test suite that includes sanity tests for dozens of scientific software packages [1]. The most recent release includes 91 scientific software packages [2]. The current model for E4S is to test release versions of packages together, and new versions are incorporated as they are released and tested. The development of E4S and its infrastructure has been transformational, yet much more can be done to find breakages in compatibility sooner than package release time.

Two ECP Software Development Kits (SDKs), those in the Math Libraries (called the xSDK [3]) and Data and Visualization areas, have stood up significant testing at the SDK level, which are subsets of the software packages comprising E4S. While testing development versions of the entire E4S right now is not practical, testing very small subsets is showing promise. Case in point, the xSDK previously tried to stand up testing for development versions of all of the packages comprising the xSDK, and that wasn't sustainable - the workflows and communication patterns necessary to deal with the errors efficiently are currently infeasible. However, attempting smaller subsets of software has been more successful. Using these small cases as building blocks for improved communication, workflows, and funding models it may be possible to extend this kind of testing to the full SDK level.

It may not make sense to try to extend the development version testing the E4S level. It depends on whether or not the increased stability and sustainability achieved at the SDK level is sufficient to promote development version workflows where needed at the E4S level. Note that using the development version of everything at the scientific software ecosystem level is likely not necessary. For example, an application may want to evaluate a change to a linear solver or a development tool, but not necessarily at the same time.

3. Conclusion

Improving the testing of the scientific software ecosystem gives users greater confidence in using scientific software packages, and makes the ecosystem and its components more maintainable for scientific software developers. Great strides have been made in recent years with E4S and the math libraries and data and visualization SDKs, but more work is necessary to improve testing robustness and efficiency and to determine more optimal workflows and policies. Fortunately, we are now in a position to learn from and build on these efforts.

References

[1] E4S 2021.11 Release Notes. https://e4s.io/talks/E4S_21.11.pdf.
[2] E4S Validation Test Suite. https://github.com/E4S-Project/testsuite.
[3] xSDK Homepage. https://xsdk.info/.