NASA/TM–2019–220456

UAS Service Supplier Checkout

How UTM Confirmed Readiness of Flight Tests with UAS Service Suppliers

Irene Skupniewicz Smith
Ames Research Center, Moffett Field, California

Joseph L. Rios
Ames Research Center, Moffett Field, California

Daniel Mulfinger
Ames Research Center, Moffett Field, California

Vijay Baskaran, SGT, Inc.
Ames Research Center, Moffett Field, California

Punam Verma, Universities Space Research Association
Ames Research Center, Moffett Field, California

December 2019

NASA/TM–2019–220456

UAS Service Supplier Checkout

How UTM Confirmed Readiness of Flight Tests with UAS Service Suppliers

Irene Skupniewicz Smith
Ames Research Center, Moffett Field, California

Joseph L. Rios
Ames Research Center, Moffett Field, California

Daniel Mulfinger
Ames Research Center, Moffett Field, California

Vijay Baskaran, SGT, Inc.
Ames Research Center, Moffett Field, California

Punam Verma, Universities Space Research Association
Ames Research Center, Moffett Field, California

National Aeronautics
and Space Administration

Ames Research Center
Moffett Field, California 94035-1000

December 2019

Introduction

NASA collaborated with industry partners to develop and test the small Unmanned Aircraft System (sUAS) Traffic Management (UTM) research platform, a software prototype used for developing airspace integration requirements for small, low altitude sUAS operations. The lessons learned from these activities will help inform the Federal Aviation Administration (FAA) on what is needed to safely manage sUAS operations. A core component of the UTM platform is the UAS Service Supplier (USS), which acts as a communications bridge between federated UTM actors to support Operators' abilities to meet the regulatory and operational requirements for UAS operations. USSs may also provide other value-added services to support UTM participants as market forces create opportunity to meet business needs.

As the UTM teams began USS flight tests, NASA quickly found that it was difficult to get a dozen independently-developed USSs functioning at comparable quality levels to ensure a level of system stability acceptable for NASA UTM flight tests. Also, NASA anticipated that the FAA would encounter similar challenges when they begin to register USSs for operational use. These realizations led to the development of USS Checkout.

USS Checkout is a set of NASA processes, and automated or semi-automated tools, designed to increase flight test efficiency as well as increase the quality of airspace management software. Since the tests are written against a specification, the USS Checkout also functions as a USS requirements test.

We tested various implementations of USS Checkouts during Technology Capability Level (TCL)-2, TCL 3 and TCL 4 flight tests [tcl2][tcl3][tcl4] with varying degrees of success.   As the USS Checkout process evolved, we learned that a good USS Checkout process is balanced for simplicity versus test coverage, and is amenable to automation. We also learned that when USS Checkout is a USS prerequisite for flight tests, flight tests were more efficient and effective.

Background

A major delivery of UTM is the USS Specification (USS Spec) [Rios-spec] which consists of a set of documents that define the expected capabilities and software requirements of a USS. Over the course of UTM, the USS Spec was extended with features novel to sUAS airspace management including negotiations, public safety and web-based authorization. Its requirements were iteratively tested with our USS partners and eventually decided by consensus across the USSs, the UTM working groups, and NASA. During this iterative process, new requirements were added, some requirements were changed and other requirements were dropped. To manage the publication of the USS Spec to our partners, NASA applied version control to the USS Spec and announced the USS Spec releases to USSs in advance of flight tests.

UTM partners gathered in NASA flight tests to demonstrate and validate UTM concepts such as negotiation, strategic deconfliction and public safety. Some partners provided USSs, and other partners provided Ground Control Software (GCS), displays and vehicles. Some flight operations used vehicle hardware, and other operations were simulated.

USSs share data which is encoded as “data models.” The process of sharing data models between two software services is called a “data exchange.” Each service implements an application programming interface (API) which contains a set of “endpoints,” at least one endpoint for each data model.

USS developers began by testing their services in isolation from other USSs. NASA's expectation was that flight test participants were thoroughly tested against the USS Spec but we soon discovered that we had a "weakest link in the chain" problem. Failure of one data exchange in a single USS could cause other data exchanges to also fail, and the entire test may be blocked and may need to be discarded, which in turn may impact the efforts of dozens of personnel, and hardware and software resources. The cost of discarding a hardware test is much higher than the cost of discarding a simulated test because a pilot may need to recheck the safety of the site, or swap a battery, creating risk of weakened or invalidated NASA research data. Thus the weakest link can jeopardize an entire scenario.

For example, a negotiation scenario is highly interdependent. In the first set of data exchanges, all USSs deconflict their proposed UTM Operations by interacting with the UTM Discovery Service. If a proposed UTM Operation intersects with another UTM Operation which has already been accepted, the proposing USS must either withdraw or negotiate. If the proposing USS wishes to negotiate, it will send a NegotiationMessage data model whereupon the other USS responds with additional data exchanges until a negotiation agreement is complete. In this set of interdependent data exchanges, all data models are validated. If one USS sends invalid data (such as a timestamp that is in the past) the entire protocol fails.

This is not to say that the USSs were built with bad quality, or with disregard to the specification. Rather, NASA’s flight tests aligned with high complexity in project management models because they had novel concepts, differentiation of participants and interdependencies between processes [Cristóbal]. USS industry partner organization were both large corporations and small companies with varying business goals and differences in available resources. As described in this document, the USS spec has hundreds of requirements, thousands of data model validations and interrelated protocols. This document describes how NASA tried to solve the weakest link problem, and summarizes the evolution of the checkout process through TCL 2, TCL 3 and TCL 4.

Checkouts for TCL 2

TCL 2 UTM did not include what is understood today as a USS; rather, the TCL 2 concepts defined a “UTM Client,” implemented by NASA partners, and a “UTM System,” directly provided by NASA. The UTM System provided deconfliction services and an Operator interface, and the UTM Client consisted of the sUAS Ground Control Station (GCS) and client software to support the Operator role.

The testing goal was to ensure a properly functioning UTM Client. The test plan was defined by a set of documents which defined the test cases which were distributed to NASA partners. The checkout was executed by each partner without the aid of NASA, as well as NASA-guided executions over telecom. For each test case, the partner recorded result data in “artifact” files. Figure 1 shows the artifact naming standard for three related Operation Rejection test cases.

```
1. UTM-IntegrationTest-{yourOrganizationName}-SimpleRejectAsyncResponse-{dateOfTest}-
v{versionNumber}.json

2. UTM-IntegrationTest-{yourOrganizationName}-SimpleRejectJsonQueryResult-{dateOfTest}-
v{versionNumber}.json

3. UTM-IntegrationTest-{yourOrganizationName}-SimpleRejectNationalPark-{dateOfTest}-
v{versionNumber}.json
```

Figure 1. TCL2 Testing Artifacts

Artifacts allowed NASA to ensure proper formatting and processing of data exchanges. Artifacts were submitted by the partner and analyzed by a UTM engineer by making queries against the UTM System’s database.

The TCL 2 checkout flow, described in Figure 2 below, is extracted from the testing documents. Note that some of the test requirements were driven by NASA’s Airworthiness Flight Safety Review Board (AFSRB) which encompassed all components of the flight test including the aircraft itself and the GCS.

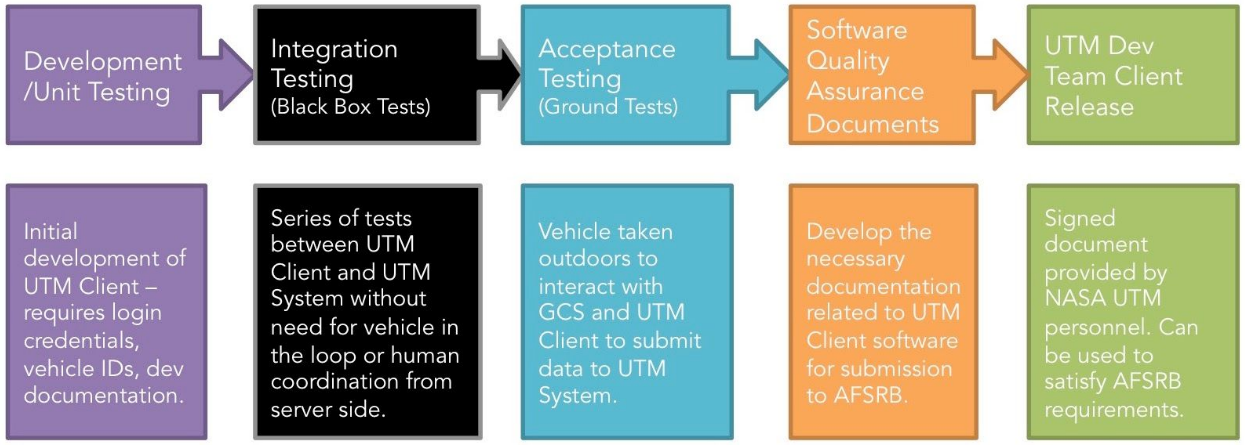

Development /Unit Testing
Integration Testing (Black Box Tests)
Acceptance Testing (Ground Tests)
Software Quality Assurance Documents
UTM Dev Team Client Release
Initial development of UTM Client – requires login credentials, vehicle IDs, dev documentation.
Series of tests between UTM Client and UTM System without need for vehicle in the loop or human coordination from server side.
Vehicle taken outdoors to interact with GCS and UTM Client to submit data to UTM System.
Develop the necessary documentation related to UTM Client software for submission to AFSRB.
Signed document provided by NASA UTM personnel. Can be used to satisfy AFSRB requirements.

Figure 2. TCL2 UAS Operator Client checkout flow

For Acceptance testing, a NASA engineer conducted scheduled telecoms with each partner. During these testing sessions, the NASA engineer would request a series of scripted and unscripted tests to be performed and verify the data received. Typical problems found centered around individual components such as time synchronization strategies, GMT time conversion, altitude units of measure, geometry or geography data models, latitude/longitude ordering, and connections to streaming data. These telecoms were necessary to prepare for effective field tests, but they were labor-intensive. In the next two years, the solutions to many of these problems would be encoded into the USS-API interface and its protocols.

TCL 3 USS Checkout using Sandbox Self-Testing

In TCL3, NASA provided a cloud-based sandbox consisting of NASA's Authorization Server, the Discovery Service and NASA's USS (NUSS). USSs were asked to use this sandbox to collaboratively checkout each others' systems over a series of NASA-defined tests. This was accomplished by having at least two other USSs attest that another USS performed a given exchange appropriately. For example, if the test involved the exchange of position information, a USS would perform position data exchanges with two other USSs. For these self-tests, the USS under test determined the testing schedule and checked the actual results with expected results. Additionally, there was a 1.5 hour human-moderated checkout process wherein the USS under test would interact via telecom with a NASA engineer. These tests were not extensive, but added further confidence in the performance of the USS implementations.

The effectiveness of the self-testing was mixed. Test milestones were difficult to coordinate across a group of USSs while simultaneously developing each USS to a common state of test-readiness. Also the sandbox did not provide a framework to create reports containing comparisons of actual data and expected data. In contrast, the human-moderated process, while labor intensive, was effective because it provided a schedule and also produced documentation of the expected data which was verified in a semi-automated way using file transfer.

The documents written for the TCL 2 and TCL 3 checkouts proved to be a great beginning point to write the scenario documents for the TCL 4 flight tests.

TCL 4 Interface Validation Tests

In TCL 4 we developed a simple and effective mechanism, “Tcl4Valtests” which validated the data exchanges and their data models in isolation from the rest of the system. Tests were scheduled to run once a day against all USSs wishing to participate in the next flight test.

Tcl4valtests were provided a number of efficiencies: they did not require set up effort by the USS, they required a minimum of NASA maintenance and oversight, and they successfully reduced the amount of shakedown time for flight tests.

As long as a USS' web service was publicly accessible, Tcl4valtests required no setup from the USS. Each test case tested a single data exchange which was initiated by NASA. Because NASA was the only party initiating data exchanges, some data models could not be tested; nevertheless, Tcl4valtest covered all but two of the sixteen models in the USS-API.

Tcl4valtests were straightforward to maintain because they were automated and USSs could diagnose most problems without help from NASA engineers. A report generator automatically produced USS-specific reports for each test execution. The report generator also maintained an aggregated history of all the test runs. Upon the completion of a test run, NASA uploaded one report to each USS-specific, cloud-hosted document folder. Report delivery was later fully automated as described in the later in this document.

As described below, NASA defined a pass-rate requirement of 100% for all participating USSs. This standard was simple to maintain because the tests were automated and the USSs could self-diagnose their problems. While NASA was careful not to share a particular USS' test results with other USSs, when a USS first achieved 100% compliance, an anonymous congratulatory announcement was posted to all the USSs. The 100% requirement kept USSs engaged and spurred USSs forward to meet their development requirements.

In the summer of 2019, the FAA used Tcl4valtest to vet participants of the FAA UTM Pilot Program [UPP]. We anticipate that, because the FAA will be registering USSs, the FAA may need a process similar to the USS Checkout process for initial registration as well as for ongoing re-verification.

Tcl4valtest Enhanced Collaborative Sprints

The rollouts of new requirements and capabilities were managed using Simulation Collaboration Sprints (Collab Sprints) [Collab Sprint 1]. Each sprint exercised a prescribed set of USS requirements and capabilities in the USS Spec. Tcl4valtests enhanced the effectiveness of the Collab Sprints because it compelled USSs to immediately implement USS Spec requirements.

For each sprint, NASA and its partners used telecoms, a chat system, and UTM Github Issues to discuss what new requirements would be tested. As per agile software methodology for sprint goals, we chose a cohesive feature set in terms of the impacted software areas. Freezes for each feature set were announced and carefully managed over the sprints. Collab Sprint end dates were not altered, rather event implementation requirements, as enforced by Tcl4valtests, were sometimes relaxed. After the tests were frozen, the USSs had two weeks to finalize their implementations and achieve a 100% pass rate.

Tcl4valtest Coverage

Each API consists of a set of endpoints and each endpoint has at least one data model that needs to be validated. For example, when Tcl4valtest, acting as a USS without elevated privileges, sends a UasVolumeReservation (UVR) to the USS under test, the data exchange should be rejected. Tcl4valtest verified that the rejection occurred and that a "forbidden" response code was returned.

Table 1 describes some of the categories needing to be tested for all USSs. In addition to the 22 data models and 25 endpoints for the basic capability set, the USS Spec defines elevated privileges capabilities, for example, the Public Safety role.

Table 1: Testable Categories

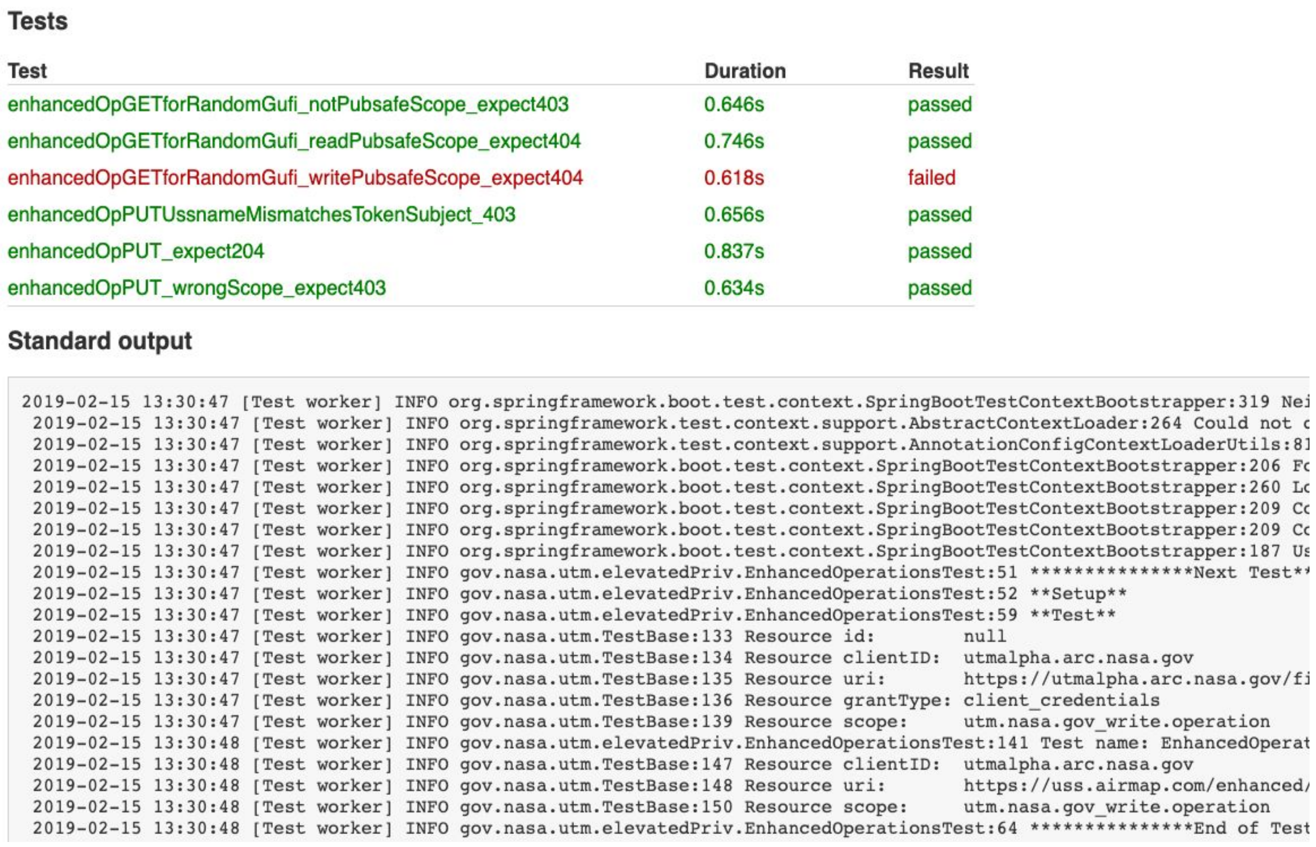
Testable Category	Count	Covered by tests	Not Covered
USS-API Data Models	22	20	Negotiation models
USS-API Endpoints	25	23	Negotiation endpoint
Public Safety Data Models	2	1	VehicleOperationData
Public Safety API Endpoints	6	1	5 GET data endpoints

Tcl4valtest Framework

The Tcl4valtest framework was hosted by Jenkins 2.0 and written using JUnit. The software team could quickly iterate through new versions of the USS-API because this specification is written in a machine-readable format (swagger 2.0) which supports code generation. Codegen allowed NASA to create new endpoints and data models, and quickly iterate through new versions of the Tcl4valtest framework.

In each test case, the test driver initiates a data exchange against the USS under test. In HTTP terminology, the test driver initiates an HTTP Request (of type GET or PUT) and validates the HTTP Response. For each data model, Tcl4valtest includes the nominal test case (the "happy path") and a set of negative test cases. For example, a negative case for HTTP PUT will construct a data model that deviates from the USS Spec so as to elicit an appropriate error response from the USS under test. The HTTP GET tests require more effort to validate because more than one response can be appropriate.

A daily test report is created by JUnit as part of the automation. USS engineers navigate this HTML report to view summaries, logs and error reports, allowing self-diagnosis. Figure 2 shows a report fragment with a summary and detail logs.



Tests

Test	Duration	Result
enhancedOpGETforRandomGufi_notPubsafeScope_expect403	0.646s	passed
enhancedOpGETforRandomGufi_readPubsafeScope_expect404	0.746s	passed
enhancedOpGETforRandomGufi_writePubsafeScope_expect404	0.618s	failed
enhancedOpPUTUssnameMismatchesTokenSubject_403	0.656s	passed
enhancedOpPUT_expect204	0.837s	passed
enhancedOpPUT_wrongScope_expect403	0.634s	passed

Standard output

```
2019-02-15 13:30:47 [Test worker] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper:319 Nei
 2019-02-15 13:30:47 [Test worker] INFO org.springframework.test.context.support.AbstractContextLoader:264 Could not c
 2019-02-15 13:30:47 [Test worker] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils:81
 2019-02-15 13:30:47 [Test worker] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper:206 Fo
 2019-02-15 13:30:47 [Test worker] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper:260 Lo
 2019-02-15 13:30:47 [Test worker] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper:209 Co
 2019-02-15 13:30:47 [Test worker] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper:209 Co
 2019-02-15 13:30:47 [Test worker] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper:187 Us
 2019-02-15 13:30:47 [Test worker] INFO gov.nasa.utm.elevatedPriv.EnhancedOperationsTest:51 ***************Next Test**
 2019-02-15 13:30:47 [Test worker] INFO gov.nasa.utm.elevatedPriv.EnhancedOperationsTest:52 **Setup**
 2019-02-15 13:30:47 [Test worker] INFO gov.nasa.utm.elevatedPriv.EnhancedOperationsTest:59 **Test**
 2019-02-15 13:30:47 [Test worker] INFO gov.nasa.utm.TestBase:133 Resource id:        null
 2019-02-15 13:30:47 [Test worker] INFO gov.nasa.utm.TestBase:134 Resource clientID:  utmalpha.arc.nasa.gov
 2019-02-15 13:30:47 [Test worker] INFO gov.nasa.utm.TestBase:135 Resource uri:       https://utmalpha.arc.nasa.gov/fi
 2019-02-15 13:30:47 [Test worker] INFO gov.nasa.utm.TestBase:136 Resource grantType: client_credentials
 2019-02-15 13:30:47 [Test worker] INFO gov.nasa.utm.TestBase:139 Resource scope:     utm.nasa.gov_write.operation
 2019-02-15 13:30:48 [Test worker] INFO gov.nasa.utm.elevatedPriv.EnhancedOperationsTest:141 Test name: EnhancedOperat
 2019-02-15 13:30:48 [Test worker] INFO gov.nasa.utm.TestBase:147 Resource clientID:  utmalpha.arc.nasa.gov
 2019-02-15 13:30:48 [Test worker] INFO gov.nasa.utm.TestBase:148 Resource uri:       https://uss.airmap.com/enhanced/
 2019-02-15 13:30:48 [Test worker] INFO gov.nasa.utm.TestBase:150 Resource scope:     utm.nasa.gov_write.operation
 2019-02-15 13:30:48 [Test worker] INFO gov.nasa.utm.elevatedPriv.EnhancedOperationsTest:64 ***************End of Test
```

Figure 2: Tcl4valtest Detailed Report

The reporting framework also generated a daily summary report which NASA posted to its internal team. With the summary reports, NASA engineers could see other noteworthy results such as a regression, or a USS achieving 100% compliance for the first time.

Tcl4valtest Results

NASA tracked its own progress using data generated from Tcl4valtests. Figure 3 shows the percentage of failures over all tests that were deployed at the time of testing. (Figure 4 will show that tests were continually deployed with each Collab Sprint leading up to the flight tests.) The X-axis represents the twelve USS participants (names not identified here). The data set contains 1377 test executions between September 6, 2018 and May 20, 2019. For all 256 days in this period, NASA delivered a browsable, detailed failure report to 12 USSs. To ensure privacy between USSs, we posted reports to USS-specific shared folders.

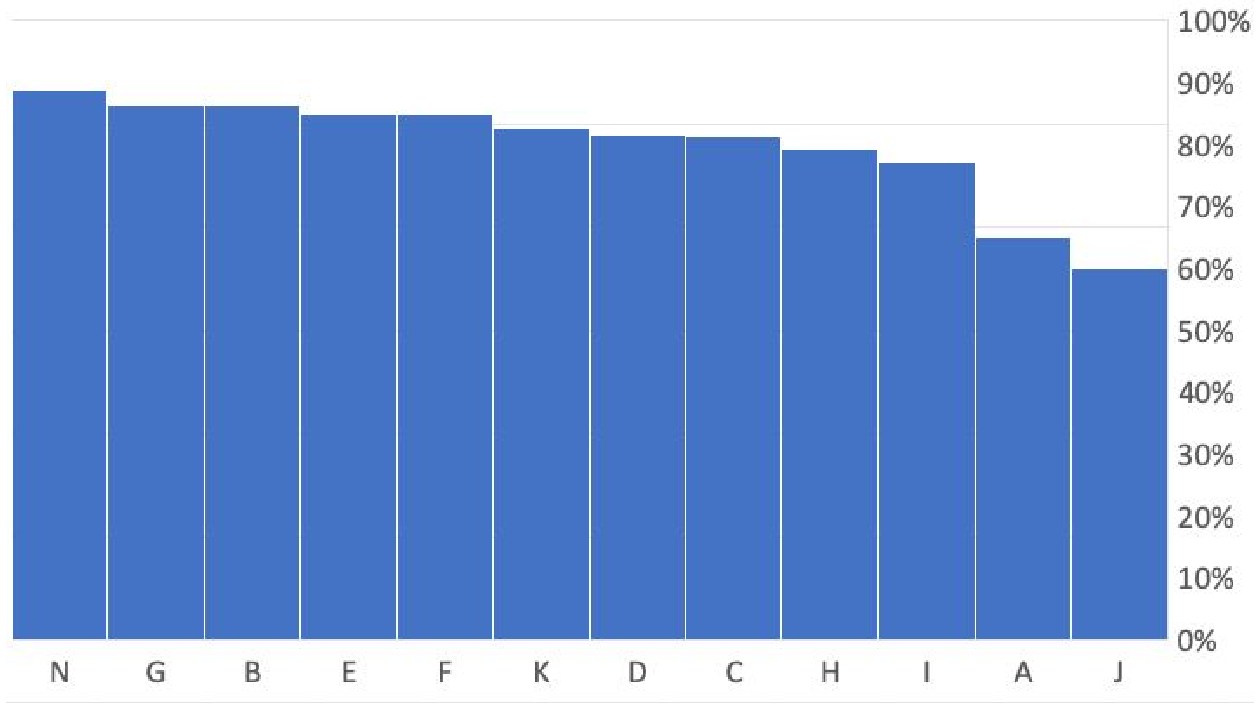


Figure 3: Per-USS Mean Score on Tcl4valtests

During the first Collab Sprint, NASA announced that there would be a test freeze and that after freeze, USSs would have two weeks to reach 100% as a prerequisite for participation in the flight tests. At the start of each Collab Sprint, NASA added new requirements and their corresponding Tcl4valtest validation tests.

Table 2 shows events influencing the test results, and Figure 4 shows the test scores as well as the increase in test coverage.

Table 2: Events Influencing Test Results

Event	Dates	Tcl4Valtest status
First two Collab Sprints	June 2018-August 2018	100% requirement announced
Collab Sprint #3	October 2018	200 tests in place
Collab Sprint #4	December 2018	344 tests in place
Freeze date	February 22, 2019	344 tests frozen
All USSs met 100%	March 6, 2019	Two weeks after freeze

TCL 4 shakedowns and flight tests for Nevada and Texas	May 15-Aug 23, 2019	Regressions

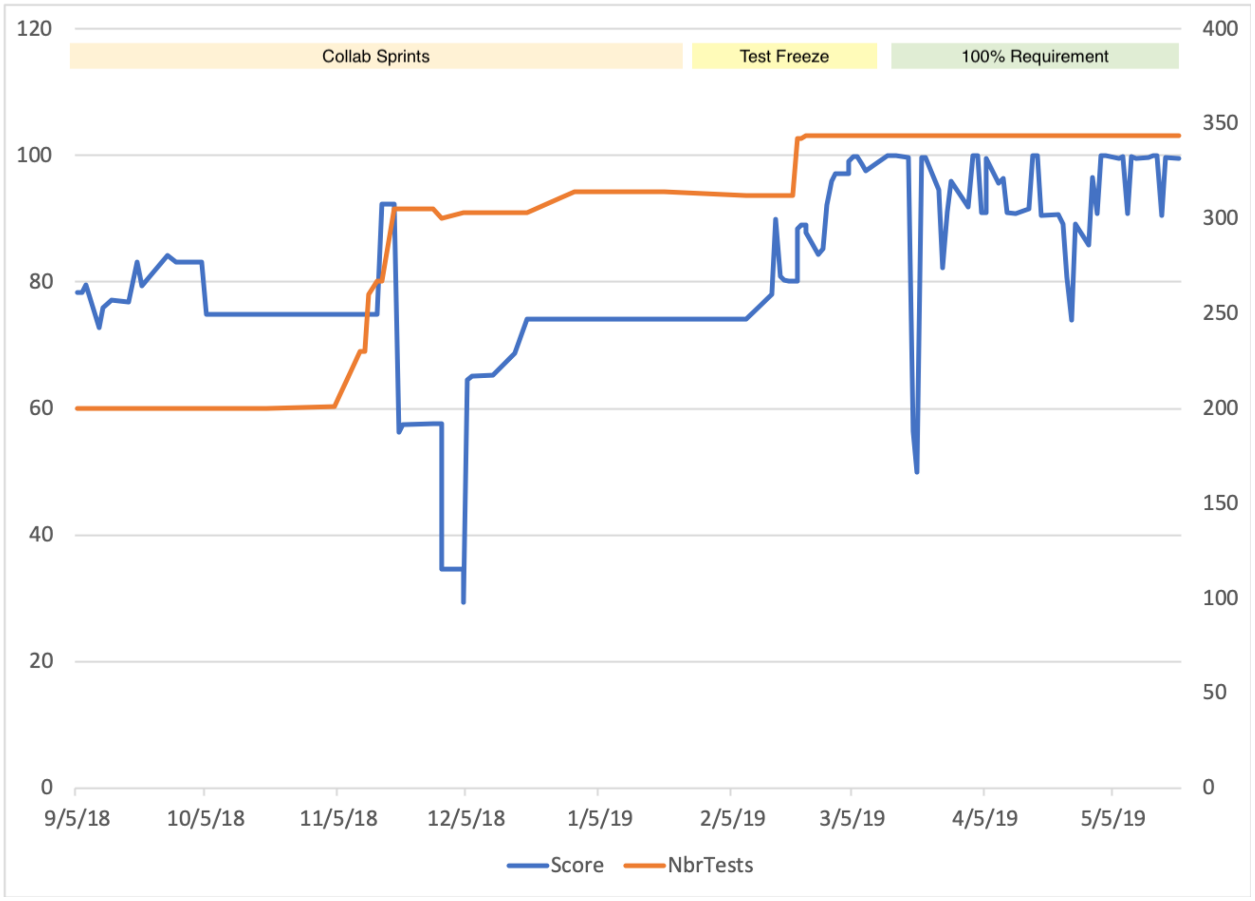


Figure 4: Daily Average Score of All USSs and Test Coverage

The blue line, Test Scores, shows that when a group of new validation tests was deployed, USS scores had a period of volatility or decline. The orange line, Number of Tests, shows that over the course of the last three sprints, test coverage increased from 200 to 344 tests.

This data shows a relationship between Collab Sprints and a subsequent period of decline. The cause of this volatility was often that USSs had not yet correctly implemented a new USS requirement. However sometimes the volatility occurred because NASA needed to adjust the requirement or the testing of the requirement. Therefore the tests were helpful in refining NASA's USS Spec.

After test freeze, USSs one by one reached 100%. However after all USSs met 100%, some USSs had intermittent failures; typically these failures were caused by USS deployment regressions. Even with 344 tests in place, some data models could not be tested. The untestable data models were those belonging to a protocol where the USS, (rather than the Tcl4valtest), initiates the HTTP request. To test these aspects we developed the Protocol Tests.

TCL 4 Protocol Tests

The primary goals of Protocol Tests were to increase test coverage relative to the Tcl4valtests, and to implement a solution that could be used by the FAA. Other goals were to enable USS-initiated, on-demand testing, to enable self-diagnosis and to minimize the effort needed to add new test cases. To discover the pros and cons of our first implementation, we built a Minimum Viable Product solution (MVP) [mvp] which was suitable for a usability study. The usability study was human-moderated using a chat system with nine USSs for three days and data was collected relative to each USS' performance.

The tests were executed in isolated geographical regions, also called "grids," creating a testing sandbox as shown in Figure 5.

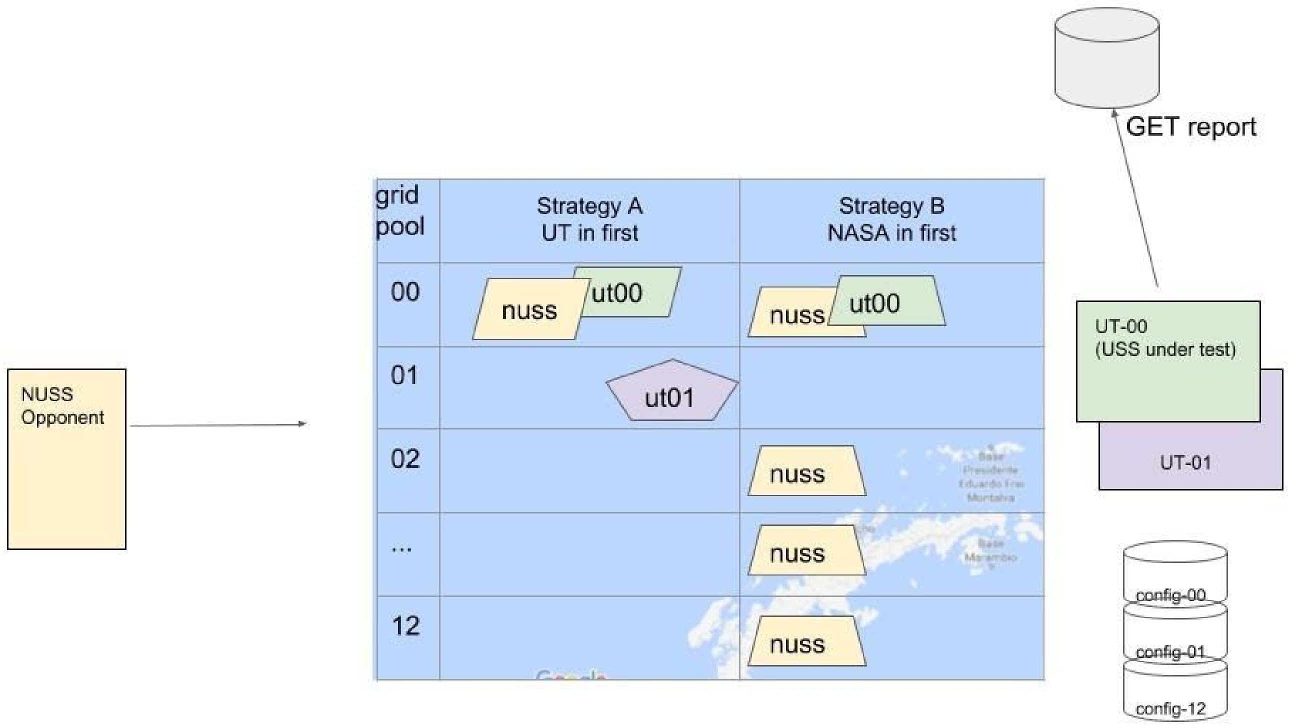


Figure 5: Protocol Test Sandbox Regions

Figure 5 illustrates the Protocol Test sandbox, consisting of grid pairs over the Antarctic Ocean. Each USS under test was assigned a single grid pair, one for each of Strategies A and B; this corresponds to one row in Figure 5. Two weeks before the first day of the usability tests, NASA produced setup kits containing USS-specific geometric data and operational parameters, and distributed the kits to each USS.

The sequence diagrams in Figures 6 and 7 below illustrates the two Protocol Test strategies, using the negotiation protocol for illustration. Strategy A verifies that the USS under test acts properly when it *receives* a negotiation request, whereas Strategy B verifies that the USS under test acts properly when it *initiates* a negotiation request.

In Figure 6 showing Strategy A, the USS under test "goes first" by planning its operation in an empty grid. In Figure 7 showing Strategy B, the NUSS opponent goes first.

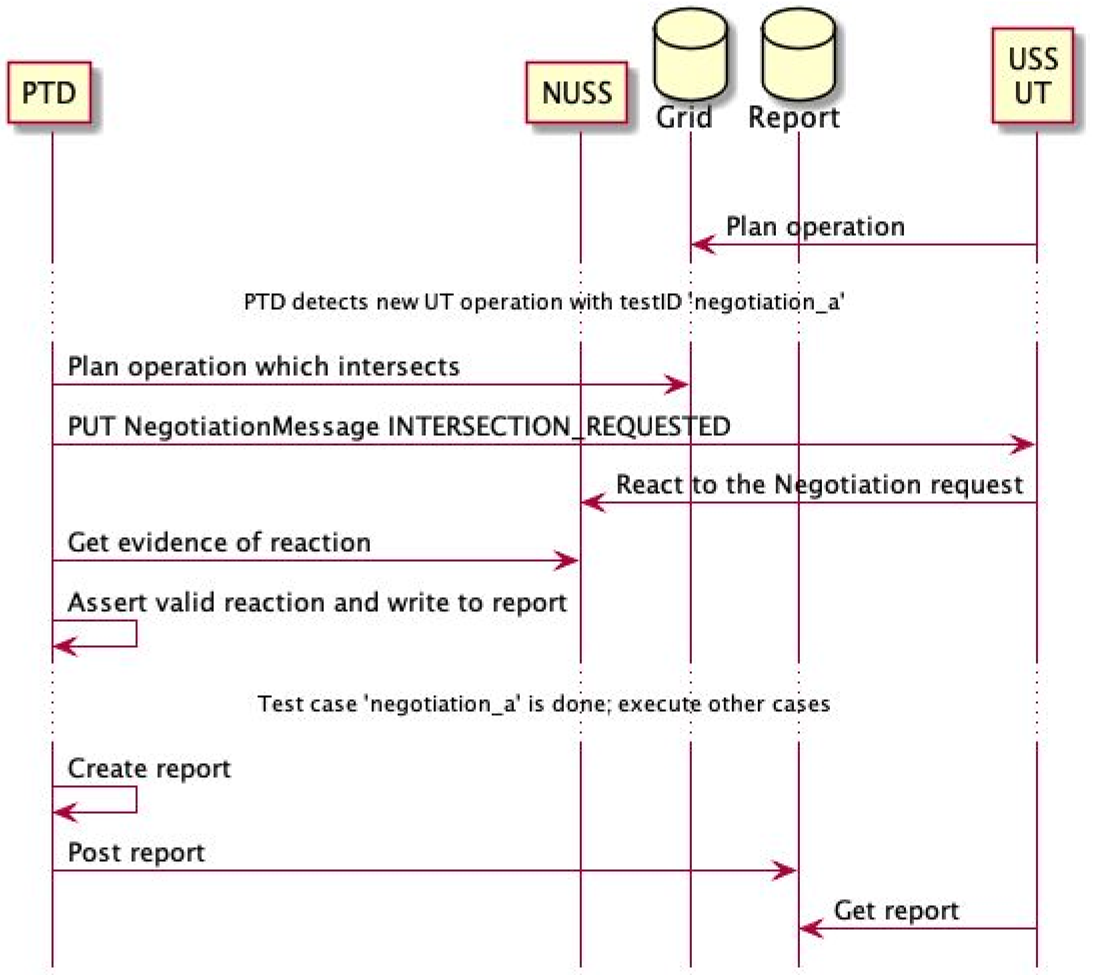

PTD
NUSS
Grid
Report
USS UT
Plan operation
PTD detects new UT operation with testID 'negotiation_a'
Plan operation which intersects
PUT NegotiationMessage INTERSECTION_REQUESTED
React to the Negotiation request
Get evidence of reaction
Assert valid reaction and write to report
Test case 'negotiation_a' is done; execute other cases
Create report
Post report
Get report

Figure 6: Verifying that a Negotiation Request is Handled using Strategy A

Figure 6 shows how NASA verifies that the USS under test acts properly when it receives a negotiation request. The actors in Figure 6 are the Protocol Test Driver (PTD), the NUSS opponent, and the USS under test. The "Grid" repository is the UTM Discovery Service which provides deconfliction services. The "Report" repository is a shared, cloud-based document store that is accessible to the USS under test.

As per Figure 6, when the USS under test plans its Operation, the PTD acts as the opponent and plans an intersecting Operation. Next, in accordance with the USS Spec, the PTD requests negotiation. Upon completion of the negotiation protocol, the PTD collects evidence of the resulting negotiation by inspecting NUSS data, and determines whether or not this test passed. Finally, the PTD generates the test report and transfers the report to the shared repository.

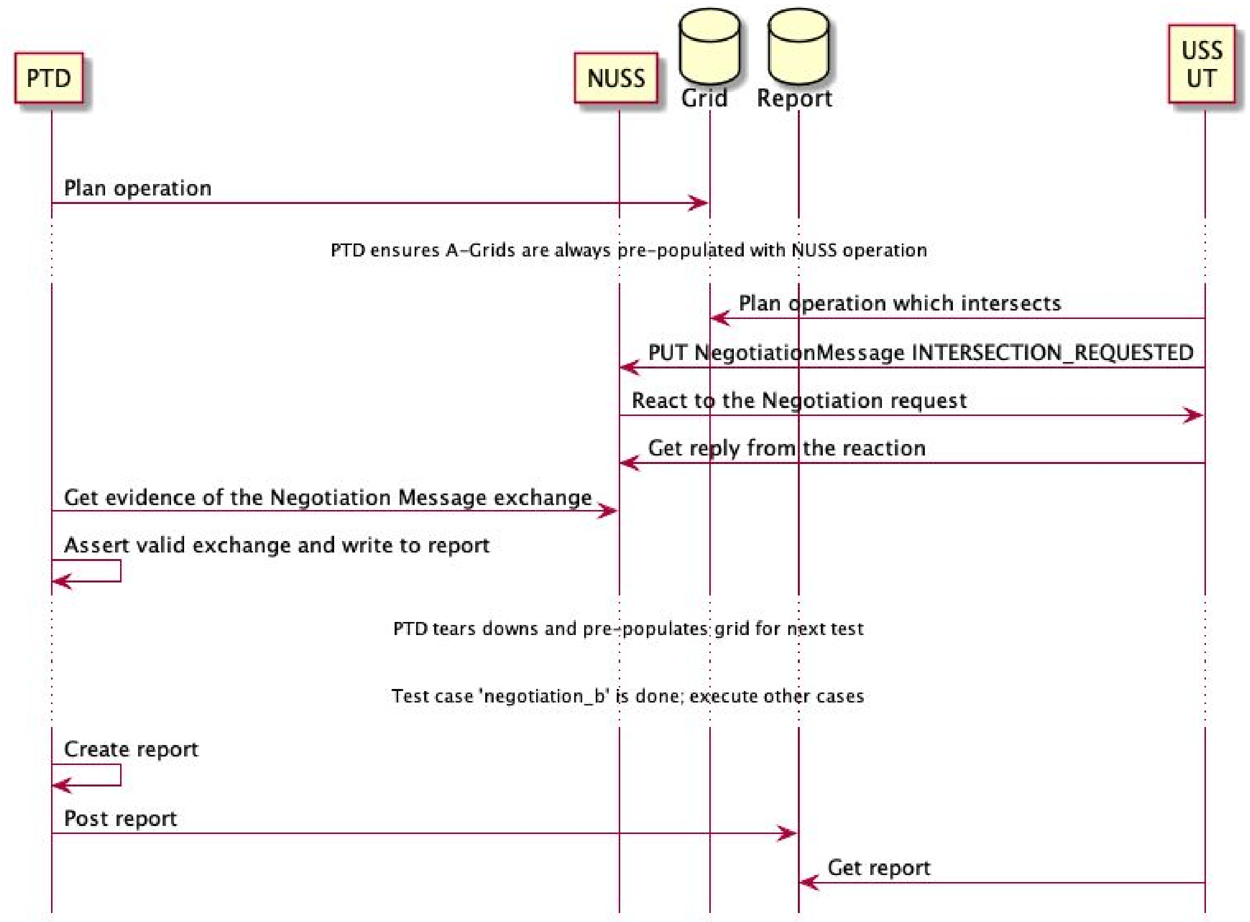

PTD
NUSS
Grid
Report
USS UT
Plan operation
PTD ensures A-Grids are always pre-populated with NUSS operation
Plan operation which intersects
PUT NegotiationMessage INTERSECTION_REQUESTED
React to the Negotiation request
Get reply from the reaction
Get evidence of the Negotiation Message exchange
Assert valid exchange and write to report
PTD tears downs and pre-populates grid for next test
Test case 'negotiation_b' is done; execute other cases
Create report
Post report
Get report

Figure 7: Testing Negotiation using Strategy B

Figure 7 shows how NASA verifies that the USS under test acts properly when it initiates a negotiation request. NUSS goes first to plan its Operation. When the USS under test is ready to initiate the negotiation test, the USS plans an Operation which intersects the NUSS Operation. The USS under test sends a negotiation request to NUSS. The PTD validates whether NUSS properly received the negotiation request and then completes the test by generating the test report.

The Protocol Test Driver

As shown in Figure 8, the Protocol Test Driver runs test cases concurrently using a pool of virtual machines.

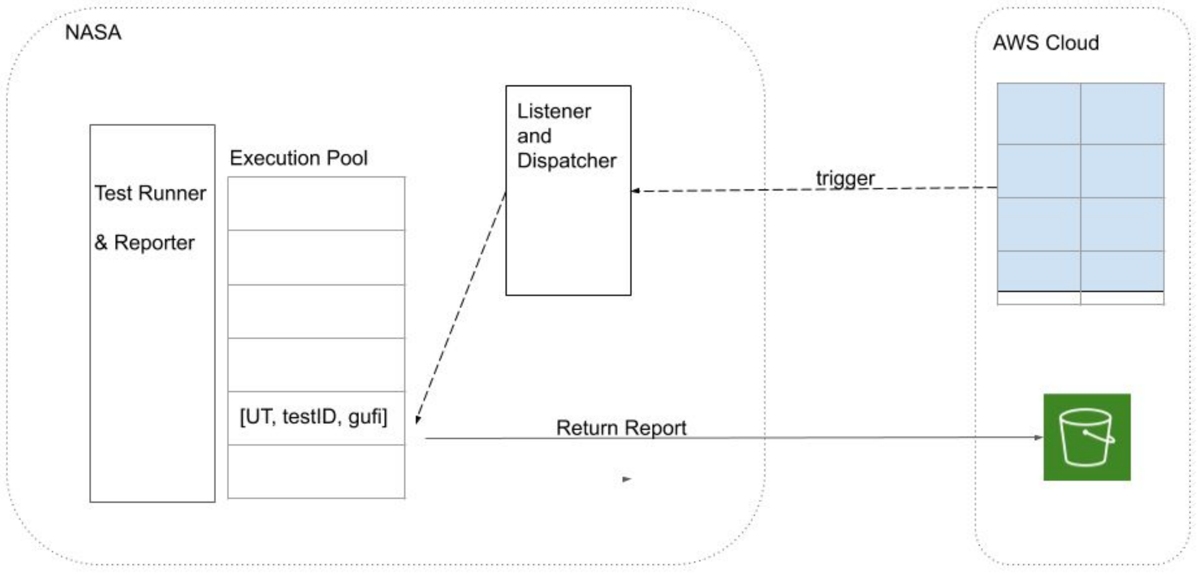


Figure 8: Protocol Test Driver

Concurrency is possible because test execution and report generation are isolated and reports are labeled uniquely using the combination of USS name, testID and Operation ID. The report history is stored in NASA's local machine as well as in the shared report repository.

After the completion of the MVP test, NASA implemented fully-automatic report delivery whereby USS-specific reports were pushed to a USS-specific AWS S3 bucket.

Results of Usability Testing

In our MVP usability testing, about 40% of the test executions resulted in automatic test validation and report generation. The Protocol Test helped partners find bugs in their USSs and their associated tooling. Because USSs were injecting their own Operations, our partners used various front end tools in addition to their USS. Engineers using these tools sometimes had usability errors such as mis-spellings and incorrect altitude units of measure.

The MVP study showed that the dual strategy approach increased test coverage, however the cost of adding a new test was increased. Going forward, if only a single setup strategy is implemented, coverage would be lower, but testing would be simpler for both NASA and the USS developer.

The MVP study demonstrated that the per-USS setup test kits worked well. Geometric flight data and operational parameters could be predefined and delivered to the USSs and USSs were able to self-diagnose their protocol failures.

During the MVP study we learned that to implement on-demand testing, the test framework needed to quickly discover a newly added Operation. The MVP implemented this mechanism using polling, because at that time NUSS did not offer a streaming endpoint. The polling implementation introduced a time lag which sometimes cause timing problems in the test suite. NUSS now has streaming endpoints that can be used for instant discovery.

Conclusion

The Tcl4valtest suite is effective and provides good test coverage, however, some data models and business logic could not be covered with this approach. The Protocol Tests provided a way to cover more models, however they required additional development to automate. The MVP study has informed follow-up requirements for tests that helps to strike the right balance between complexity and coverage.

The UTM Project expects that the FAA will be involved in some way to approve or “register” USSs whereby the FAA (or an entity acting on its behalf) can evaluate a USS’ capabilities. In addition to the basic capabilities of a USS, the FAA may want to vet enhanced USS capabilities such as the USS Public Safety role. A test process providing even partial coverage provides abundant, hands-on evidence related to USS capabilities. Moreover, these test suites create a history of USS capability data.

References

[Collab Sprint 1] Rios, J., Smith, I., Venkatesen, P., Smith, D., Baskaran, V., Jurcak, S., Strauss, R., Iyer, S., Verma, P., “UTM UAS Service Supplier Development: Sprint 1 Toward Technical Capability Level 4”, NASA Technical Memorandum, NASA/TM-2018-220024, November 2018, <https://utm.arc.nasa.gov/docs/UTM_UAS_TCL4_Sprint1_Report.pdf>.

[Collab Sprint 2] Rios, J., Smith, I., Venkatesen, P., Smith, D., Baskaran, V., Jurcak, S., Iyer, S., Verma, P., “UTM UAS Service Supplier Development: Sprint 2 Toward Technical Capability Level 4”, NASA Technical Memorandum, NASA/TM-2018-220050, December 2018, <https://utm.arc.nasa.gov/docs/2018-UTM_UAS_TCL4_Sprint2_Report_v2.pdf>.

[Cristóbal] San Cristóbal, J., “Complexity in Project Management”, Procedia Computer Science, November 2017, <https://www.sciencedirect.com/science/article/pii/S1877050917323001>.

[mvp] Agile Alliance, Minimum Viable Product, <https://www.agilealliance.org/glossary/mvp/#q=~(infinite~false~filters~(tags~(~'mvp))~searchTerm~'~sort~false~sortDirection~'asc~page~1)>.

[North] North, D., “Faster Organizations, Faster Software”, 2006,<https://dannorth.net/introducing-bdd/>.

[Rios-security] Rios, J., Smith, I., Venkatesen, P., “UTM Authentication and Authorization Framework”, NASA Technical Memorandum, NASA/TM-2019-20364, September 2019.

[Rios-spec] Rios, J., et al., “UAS Service Supplier Specification,” NASA Technical Memorandum, NASA/TM-2019-220376, October 2019.

[tlc2] UAS Technical Capability Level 2 Unmanned Aircraft System Traffic Management (UTM) Flight Demonstration: Description and Analysis, Homola, J., Christoph M., Dao, Q., Claudatos, L., Martin,L., Mercer, J., IEEE-DASC-September 17-21, 2017 St. Petersburg, FL.

[tcl3] Flight Demonstration of Unmanned Aircraft System (UAS) Traffic Management (UTM) at Technical Capability Level 3, Aweiss, A., J. Homola J., Rios, J., Jung, J., Johnson, M., Mercer, J., Modi, H., Torres, E., IEEE-DASC, September 8-12, 2019, San Diego, CA.

[tcl4] Flight Demonstration of Unmanned Aircraft System (UAS) Traffic Management (UTM) at Technical Capability Level 4, not yet published.

[UPP] Federal Aviation Administration, “UTM Pilot Program”, 2018, <https://www.faa.gov/uas/research_development/traffic_management/utm_pilot_program/>.