INL RPT-24-04645

FORCE Regression Testing

June / 2024

Botros N. Hanna
Idaho National Laboratory

Dylan J. McDowell
Idaho National Laboratory

Joshua J. Cogliati
Idaho National Laboratory

Paul Talbot
Idaho National Laboratory

IES
Integrated Energy Systems

INL RPT-24-04645

FORCE Regression Testing

Botros N. Hanna
Dylan J. McDowell
Joshua J. Cogliati
Paul Talbot

June 2024

Idaho National Laboratory
Integrated Energy Systems
Idaho Falls, Idaho 83415

http://www.ies.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

Page intentionally left blank

EXECUTIVE SUMMARY

Through programs such as the Light Water Reactor Sustainability and Integrated Energy Systems (IES), the U.S. Department of Energy has invested in the Framework for Optimization of ResourCes and Economics (FORCE) software framework, which is used (Idaho National Laboratory 2024a) for the technical and economic analysis of nuclear integrated energy systems. Nuclear IES expands the use of nuclear, from traditional baseload electricity generation to a flexible and adaptive source of combined heat and power. Nuclear heat can be used in the production of a variety of energy currencies, such as hydrogen and ammonia, as well as in other heat applications, including water desalination and district heating.

FORCE is designed to provide interconnected analysis tools that enable the accurate technical and economic assessment of specific nuclear IES configurations for individual energy markets. FORCE consists of three main analysis pathways: HYBRID (Idaho National Laboratory 2024b), which contains high-resolution physical models for IES; Holistic Energy Resource Optimization Network (HERON) (Idaho National Laboratory 2024c), which analyzes the long-term economic viability of IES; and Optimization of Real-time Capacity Allocation (ORCA) (Idaho National Laboratory 2024d), designed for real-time control of IES via digital twins and optimal decision making, including autonomous and remote operation research.

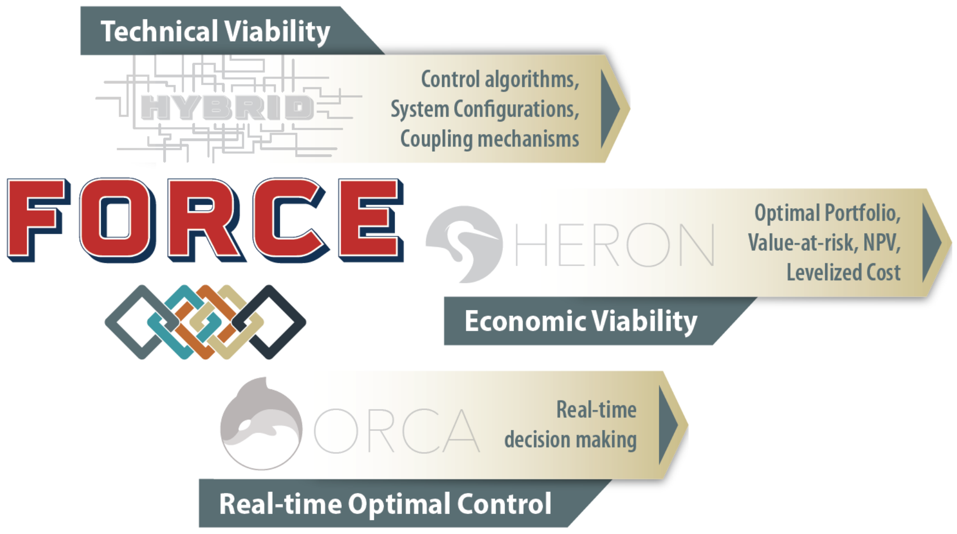

Technical Viability
HYBRID
Control algorithms, System Configurations, Coupling mechanisms
FORCE
HERON
Optimal Portfolio, Value-at-risk, NPV, Levelized Cost
Economic Viability
ORCA
Real-time decision making
Real-time Optimal Control

Figure ES1. Major pathways of FORCE (HYBRID, HERON, ORCA).

The development of the FORCE ecosystem is guided by three pillars: capability, which assures that the computational requirements of IES analysis are met by the software tools; reliability, which provides for consistent code performance and expected behaviors; and accessibility, which lowers the barrier to entry for using the software and accelerates analysis by users beyond FORCE’s primary developers.

The reliability of the FORCE ecosystem is established according to the American Nuclear Society's Nuclear Quality Assurance (NQA-1) program (American Society of Mechanical Engineers 1982), . Specific levels of software quality assurance within NQA-1 are applied to each software tool in FORCE. As the tools within FORCE have matured, some integration algorithms for accurately connecting the software tools for holistic analysis have been developed and deployed within the FORCE software repository. In accordance with NQA-1 standards, regression tests are required to guarantee the software performs consistently even when new capabilities are added to the software.

In this report, we document the deployment of both unit tests, which test the consistent behavior of small pieces of the FORCE code base, as well as integration tests, which test the consistent performance of full-use cases for the FORCE integration algorithms. We further document the encapsulation of these tests within a test harness, which collectively checks for each successful test completion on demand (see Figure ES2). Finally, we document the automation of the test harness using GitHub actions (GitHub 2024), which require that all tests succeed before any new capabilities can be added—or changes be made—to the FORCE integration software.

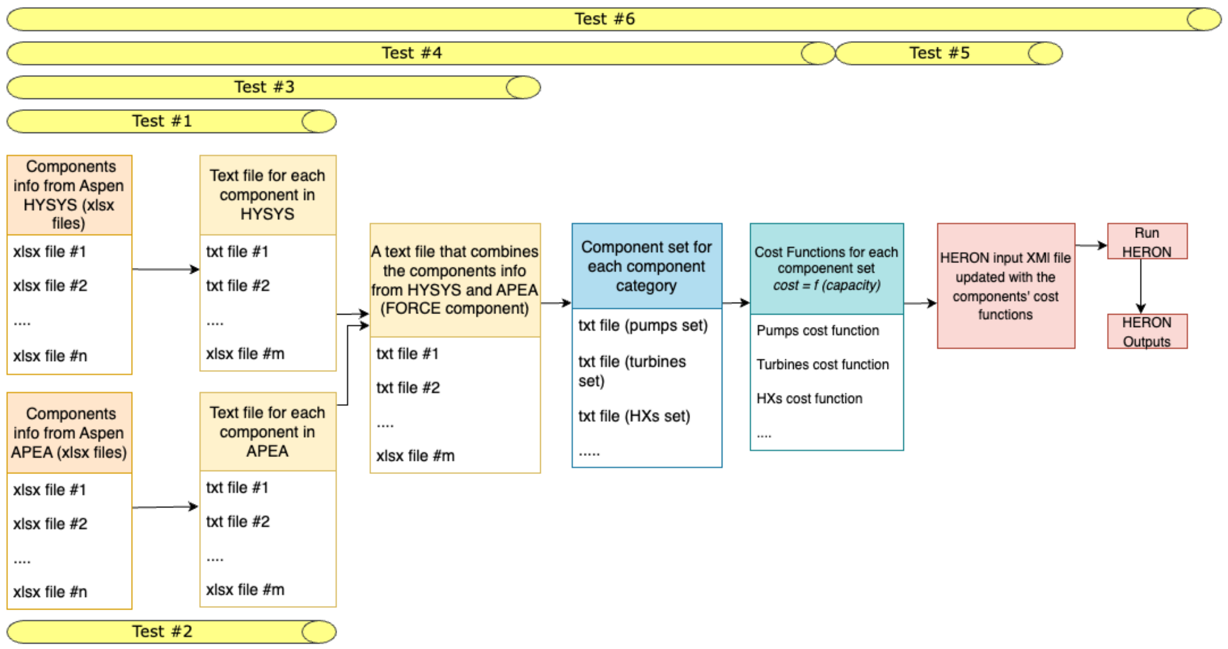


Figure ES2. Testing the integration of Aspen HYSYS–APEA and HERON.

Page intentionally left blank

CONTENTS

FIGURES

TABLES

Page intentionally left blank

ACRONYMS

APEA	Aspen Process Economic Analyzer
FORCE	Framework for Optimization of ResourCes and Economics
HERON	Holistic Energy Resource Optimization Network
IES	integrated energy systems
INL	Idaho National Laboratory
ORCA	Optimization of Real-time Capacity Allocation
RAVEN	Risk Analysis Virtual Environment
TES	thermal energy storage

Page intentionally left blank

1. INTRODUCTION

To analyze the technical and economic potential of integrated energy systems (IES), the Framework for Optimization of ResourCes and Economics (FORCE) tool suite was developed through a collaboration between national laboratories. FORCE can be used to analyze various types of nuclear reactors, renewable technologies, and energy markets, as well as the technical and economic viability of myriad IES configurations.

FORCE includes several codes that individually answer a portion of a problem (see Figure). These codes include:

- The Holistic Energy Resource Optimization Network (HERON), which provides algorithms for analyzing the long-term viability of potential IES technologies
- HYBRID, which includes several transient process models that represent the physical dynamics of interest of various IES technologies:
    - Transient process models developed in the Modelica language
    - Aspen HYSYS IES steady-state models
    - Aspen Process Economics Analyzer (APEA) cost models for IES components
- Optimization of Real-time Capacity Allocation (ORCA)
- Feasible actuator range modifier, which predicts the system state in the future
- Agent-based capacity expansion mode.

The continued maturing of the FORCE tool suite requires further interconnections between the various tools in the suite to ensure consistent analysis. The vertical integration between the tools used in the IES analysis aims to:

1. Unify the different IES/FORCE tools inside one environment
2. Automate the data transfer between different codes to accelerate the IES techno-economic analysis.

Integrating IES tools facilitates the sensitivity analysis studies and quantifying the uncertainty propagation through IES codes. Previous work (Saeed 2022a, Hanna 2023) in this area prioritized automating the data transfer processes that were necessary for recent IES studies (Saeed et al. 2022b, Wendt et al. 2022). Conducting these studies include:

- Data transfer between Modelica and HERON
- Integrating FORCE tools with Aspen HYSYS and APEA. These tools are not part of the Idaho National Laboratory (INL) FORCE tools but are frequently used by IES analysts.

A thermal energy storage (TES) use case emphasized the need to integrate IES tools (Saeed et al. 2022b). The modeling and data transfer between the IES codes in this use case are represented in Figure and summarized here:

- Designing a balance of plant model of several reactor designs using Aspen HYSYS
- Estimating the components' costs via APEA
- Grouping all the components into three separate groups: "charge," "discharge," and "storage", based on whether they produce, consume, or store energy
- Developing cost functions for each component group (solving a curve-fitting problem)

- Implementing the cost functions in the HERON input file
- Creating the reduced order autoregressive moving-average mode (ARMA) model of the market data (arma.pk file)
- Running HERON to calculate the optimized components' capacities and the optimized energy dispatch
- Running HYBRID to analyze the dynamic behavior of the IES of interest and to evaluate system controls
- Checking whether the Aspen HYSYS and HYBRID models are consistent (comparing the quasi-steady-state mode in HYBRID with the Aspen HYSYS model).

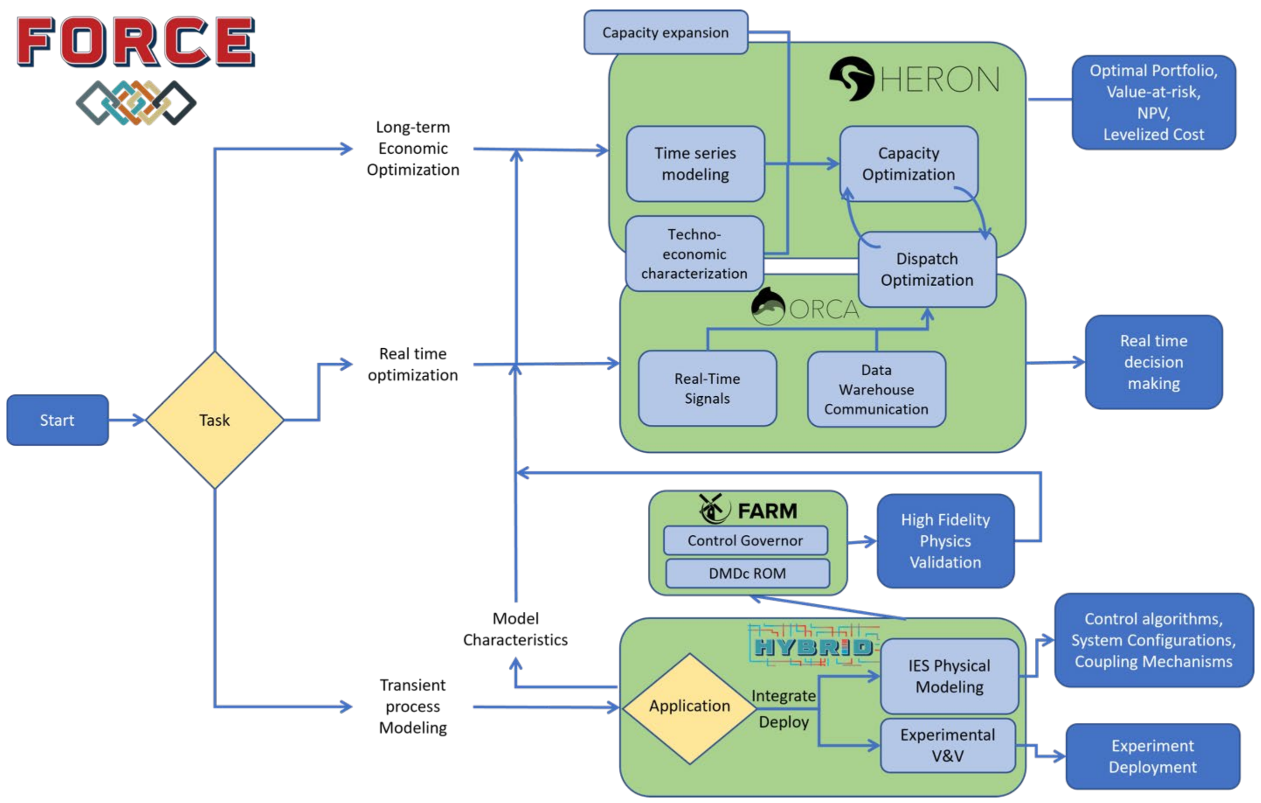

FORCE
Capacity expansion
HERON
Optimal Portfolio, Value-at-risk, NPV, Levelized Cost
Long-term Economic Optimization
Time series modeling
Capacity Optimization
Techno-economic characterization
Dispatch Optimization
ORCA
Real time optimization
Real-Time Signals
Data Warehouse Communication
Real time decision making
Start
Task
FARM
Control Governor
DMDc ROM
High Fidelity Physics Validation
Model Characteristics
HYBRID
Control algorithms, System Configurations, Coupling Mechanisms
Transient process Modeling
Application
Integrate
Deploy
IES Physical Modeling
Experimental V&V
Experiment Deployment

Figure 1. FORCE includes several tools for several tasks.

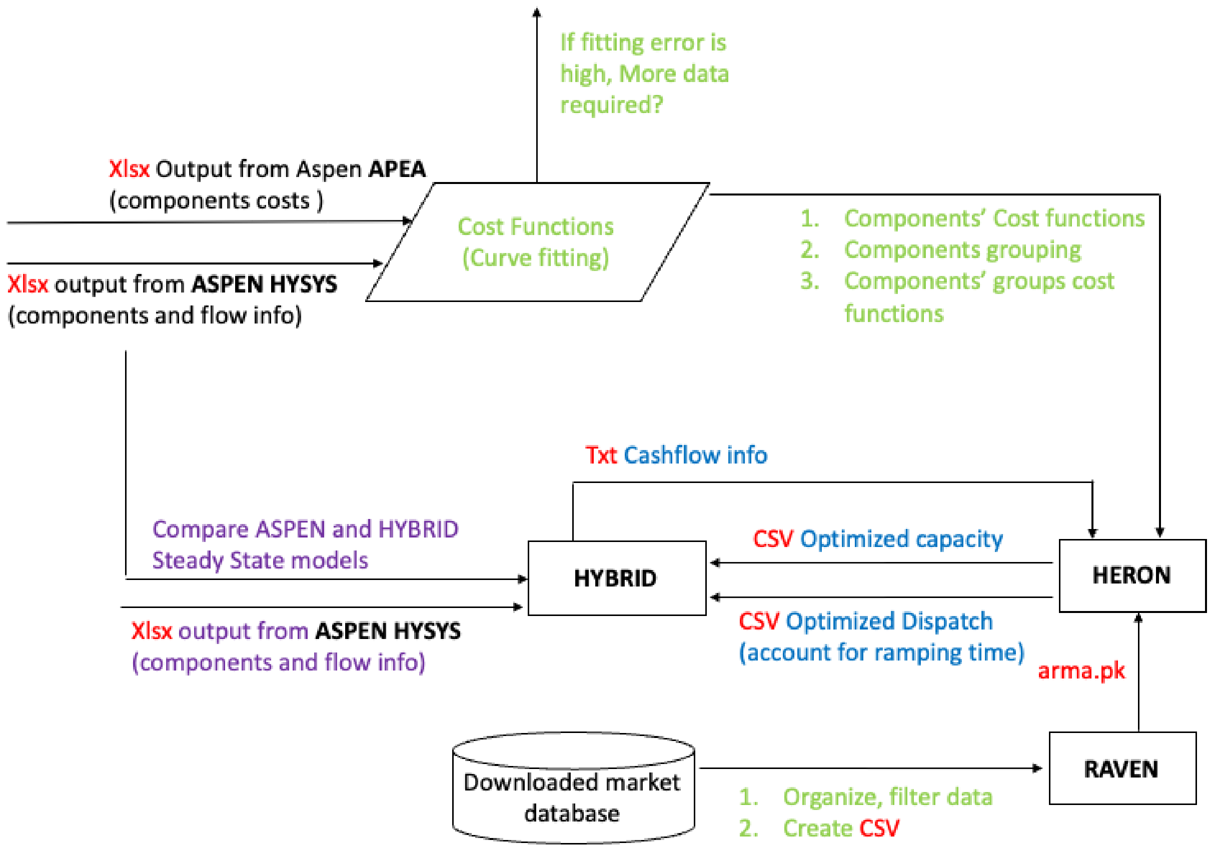

If fitting error is high, More data required?
Xlsx Output from Aspen APEA (components costs)
Xlsx output from ASPEN HYSYS (components and flow info)
Cost Functions (Curve fitting)
1. Components' Cost functions
2. Components grouping
3. Components' groups cost functions
Txt Cashflow info
Compare ASPEN and HYBRID Steady State models
HYBRID
CSV Optimized capacity
HERON
CSV Optimized Dispatch (account for ramping time)
Xlsx output from ASPEN HYSYS (components and flow info)
arma.pk
Downloaded market database
1. Organize, filter data
2. Create CSV
RAVEN

Figure 2. Data transfer between several IES codes (in the TES use case) requires vertical integration.

Previous work (Saeed 2022a, Hanna 2023) in the vertical integration of FORCE tools can be summarized as follows:

- Developing a code that integrates Aspen HYSYS, APEA, and HERON by automating the data transfer between them (Since this integration requires developing cost functions of different components, scripts that produce cost functions were included too.)
- Developing the software structure that enables integrating other IES codes, if needed
- Autoloading the components' economic info from HYBRID to HERON
- Autoloading the optimized energy dispatches from HERON to HYBRID.

Eventually, FORCE will be a central hub for the codes used in IES analyses (Figure 3), which will facilitate communication between these codes. Also, all the components' info from all IES analyses will be stored in one place for future analyses.

The scope of this work is the regression testing of the vertical integration algorithms that have been implemented in FORCE. *Regression testing* refers to running tests to ensure that software works as intended after any code changes, updates, or revisions. Regression tests are required to guarantee the software performs consistently even when new capabilities are added to it. These regression tests are necessary to establish the reliability of the FORCE ecosystem.

In this report we document the deployment of integration tests (Section 2), which tested the consistent performance of full-use cases for the FORCE integration algorithms, and unit tests (section 3), which tested the consistent behavior of small pieces of the FORCE code base. We also document the automation of the test harness (Section 4) using GitHub actions (GitHub 2024) to ensure that all the regression tests succeed before any new capability or other changes can be added to the FORCE integration software.

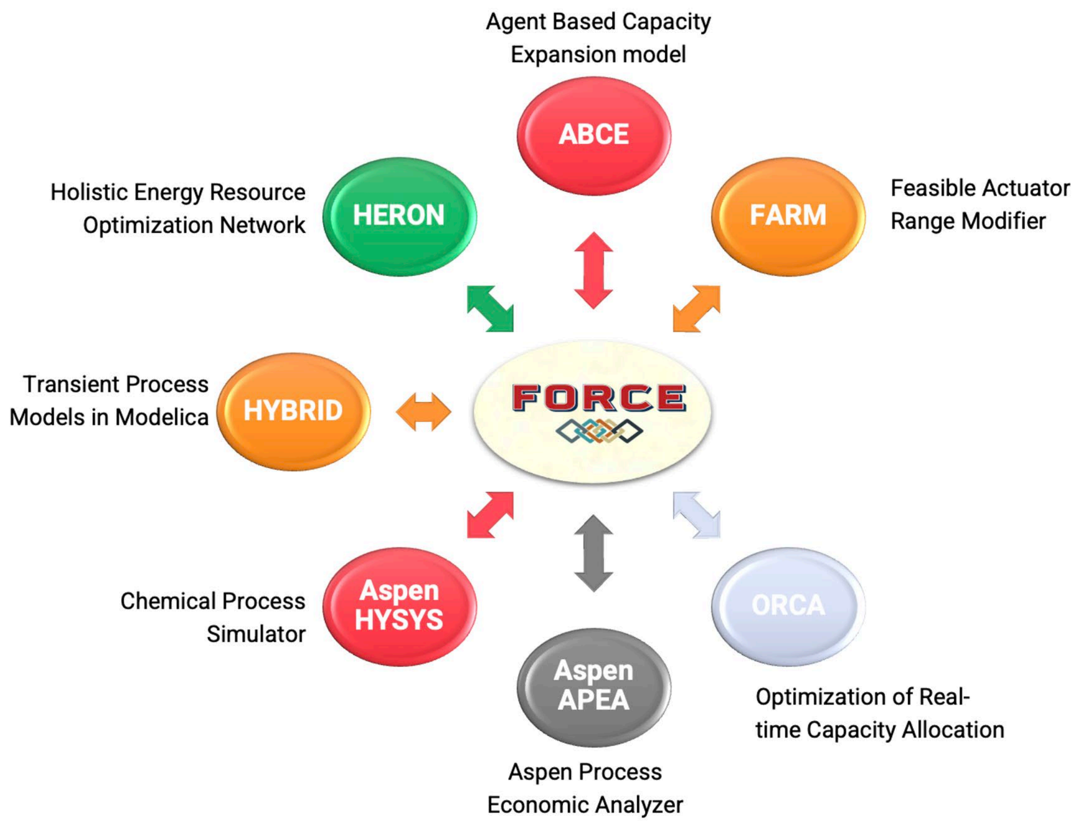

Agent Based Capacity Expansion model
ABCE
Holistic Energy Resource Optimization Network
HERON
FARM
Feasible Actuator Range Modifier
Transient Process Models in Modelica
HYBRID
FORCE
Chemical Process Simulator
Aspen HYSYS
ORCA
Aspen APEA
Optimization of Real-time Capacity Allocation
Aspen Process Economic Analyzer

Figure 3. FORCE will serve as a central hub for the codes used in IES analyses.

2. INTEGRATION TESTING

Several tests have been implemented to test the vertical integration between Aspen HYSYS, APEA, and HERON. Aspen HYSYS and APEA are not part of the INL FORCE tool suite, but they are frequently used in IES analyses. Cost data about several components (pumps, turbines, etc.) from APEA, and the components specs from Aspen HYSYS, are used to create cost functions, which HERON uses to conduct capacity and energy dispatch optimization. This Aspen-HERON integration testing is demonstrated in Figure 4.

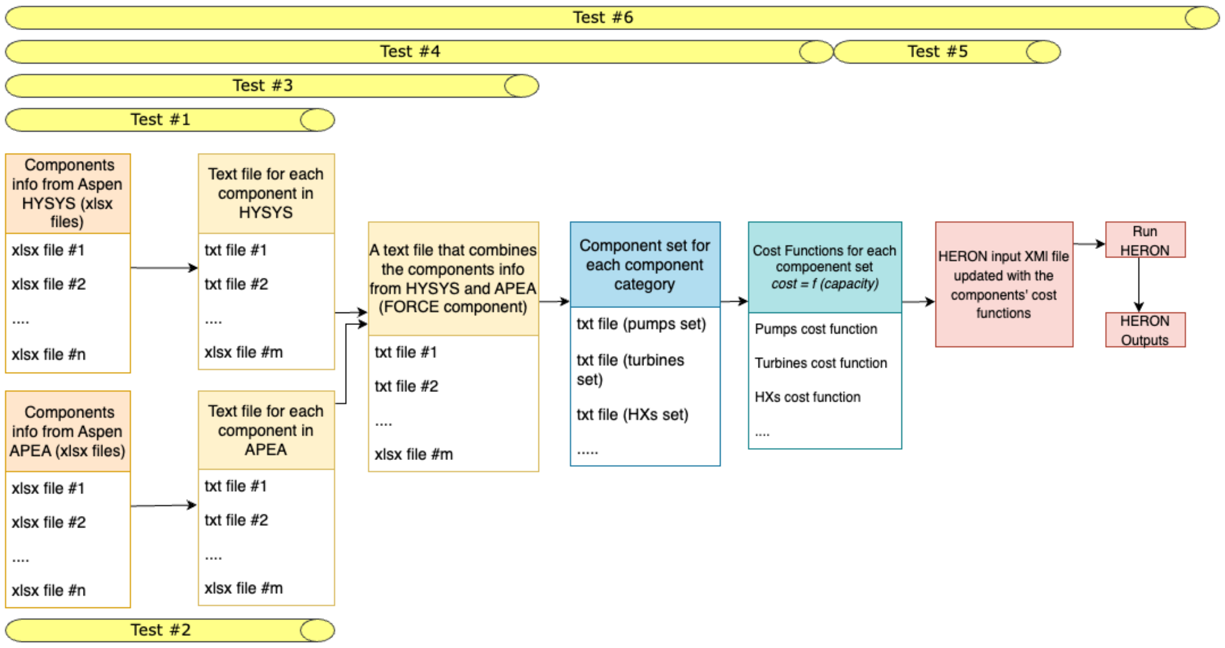
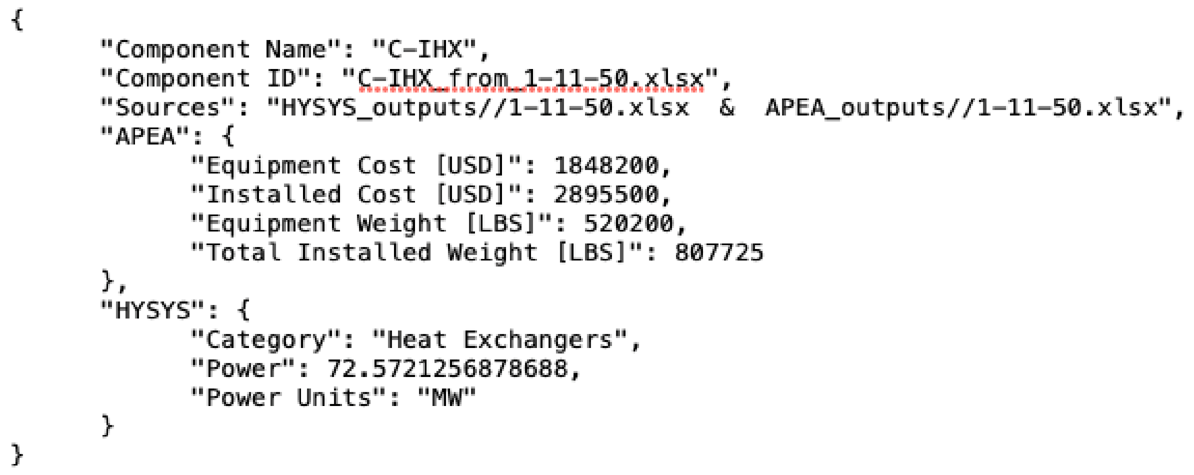


Figure 4. Integration tests (Aspen HYSYS–APEA and HERON).

In Figure 4, test #1 and test #2 cover the process of extracting the components specs (capacities, costs, etc.) from the output sheets (simulation results of Aspen HYSYS and APEA) and of storing the specs of each component in text files. Test #3 covers these tests as well as the process of combining/matching the component info from Aspen HYSYS and APEA. This results in “FORCE components”—that is, any components with economic info imported from several codes. Figure 5 is an example of a FORCE component.

Test #4 covers test #3 and leverages the components info to create cost functions for each component (see Figure for examples of cost functions). Test #5 covers the transfer of the cost functions to the HERON input file. Test #6 is an overall test that covers all five tests and runs the newly created HERON input file (see Figure 7). If the test#6 failed, it would be relatively easy to determine which piece of the code has failed by running the other tests (from #1 to #5) that cover specific parts of the code.

Table 1 summarizes the inputs, outputs, and testing criteria for each test.



```
{
        "Component Name": "C-IHX",
        "Component ID": "C-IHX_from_1-11-50.xlsx",
        "Sources": "HYSYS_outputs//1-11-50.xlsx  &  APEA_outputs//1-11-50.xlsx",
        "APEA": {
              "Equipment Cost [USD]": 1848200,
              "Installed Cost [USD]": 2895500,
              "Equipment Weight [LBS]": 520200,
              "Total Installed Weight [LBS]": 807725
        },
        "HYSYS": {
              "Category": "Heat Exchangers",
              "Power": 72.5721256878688,
              "Power Units": "MW"
        }
}
```

Figure 5. A FORCE component.

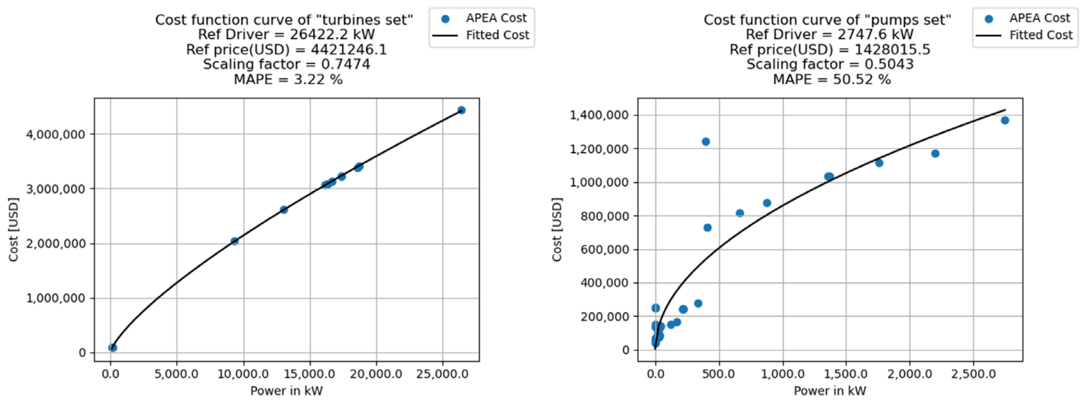


Figure 6. Cost functions.

```xml
<Component name="pumps set">
 <!-- This component info are imported from: Sets1//componentSet_pumps.txt-->
 <economics>
  <CashFlow name="pumps set_capex" type="one-time" taxable="True" inflation="None" mult_target="False">
   <!-- Default values are assigned to the cashflow parameters and need to be reviewed by the user-->
   <reference_driver>
    <fixed_value>2.74763868640817</fixed_value>
    <!--Units : MW-->
   </reference_driver>
   <reference_price>
    <fixed_value>-1428015.4597306298</fixed_value>
    <!--Reference Price (USD)-->
   </reference_price>
   <scaling_factor_x>
    <fixed_value>0.5042733812345344</fixed_value>
   </scaling_factor_x>
   <!--Note that the cost function curve fitting error is 50.52 %-->
  </CashFlow>
 </economics>
</Component>

<Component name="HX set">
 <!-- This component info are imported from: Sets1//componentSet_heat_exchangers.txt-->
 <economics>
  <CashFlow name="HX set_capex" type="one-time" taxable="True" inflation="None" mult_target="False">
   <!-- Default values are assigned to the cashflow parameters and need to be reviewed by the user-->
   <reference_driver>
    <fixed_value>11.637680295019</fixed_value>
    <!--Units : MW-->
   </reference_driver>
   <reference_price>
    <fixed_value>-530144.6197970972</fixed_value>
    <!--Reference Price (USD)-->
   </reference_price>
   <scaling_factor_x>
    <fixed_value>0.46533781554473563</fixed_value>
   </scaling_factor_x>
   <!--Note that the cost function curve fitting error is 22.24 %-->
  </CashFlow>
 </economics>
</Component>
```
Figure 7. A HERON input file updated with cost functions using info from Aspen HYSYS and APEA.

Table 1. A summary of the regression tests.

#	Inputs	Outputs	Testing Criteria
1	Excel sheets from Aspen HYSYS (component info)	Relevant components info stored in text files	The similarity between the output text files and the reference text files
2	Excel sheets from APEA (component info)	Relevant components info stored in text files	The similarity between the output text files and the reference text files
3	Excel sheets from Aspen HYSYS and APEA (component info)	Merged component info from HYSYS and APEA in text files	The similarity between the output text files and the reference text files
4	Excel sheets from Aspen HYSYS and APEA (component info) and a user-defined file for which components need to be grouped	Component cost functions	The similarity between the output cost functions and the reference cost functions
5	A user-defined file for which components need to be grouped, and an initial HERON input file	A new HERON input file updated with cost functions	The similarity between the HERON input file and the expected HERON input file
6	Excel sheets from Aspen HYSYS and APEA (component info), a user-defined file for which components need to be grouped, and an initial HERON input file	Merged component info from HYSYS and APEA in text files, component cost functions, a new HERON input file updated with cost functions, and the HERON output (CSV file)	- The similarity between the output text files and the reference text files - The similarity between the output cost functions and the reference cost functions - The similarity between the HERON input file and the expected HERON input file The similarity between the HERON output CSV file and the expected HERON output CSV file

3. UNIT TESTING

Unit testing involves testing individual components or units of a software application to ensure that each part functions correctly. A unit is the smallest testable part of an application, often a single function, method, or class. Implementing unit tests allows for automated verification that the code is working as expected. Since FORCE follows a continuous integration release process, these tests are integral to the quality assurance encouraged by INL's software distribution team. Not only is it important to deliver working code, but also to monitor how the code changes over time with respect to its performance, output, and stability.

FORCE unit testing has specific requirements that address its unique capabilities. FORCE is currently organized such that each code that interoperates with FORCE has its own Python module that contains the logic for communicating data between the respective software. This organization breaks the code down into modules that can be individually tested. Since FORCE's main capability is taking in many different forms of data as input and outputting results to many different formats, we needed to construct tests that (1) don does not rely on external dependencies and (2) do ensure the code operates as it should.

In other words, a unit test should not need to run external software to get the input data required by FORCE. It also should not need to run other software to get output data. If the unit tests relied on external operations, they would not truly be testing the software because external software can introduce many different errors and bugs. This led us to implementing unit tests using a "mock-up" method that is ubiquitous in software engineering. This method allows us to construct the input/output data we expect from FORCE without relying on additional methods of creation.

In addition to the tests themselves, we implemented the infrastructure for automating the tests as well. Our tests relied on the native Python module "unittest" (Python Software Foundation, 2024). This module allows the testing logic to be written and stored in several files within the "tests" folder of the FORCE repository. When the tests are run, all the files within the folder are run and tested, resulting in a summary output indicating pass/failure results for each test.

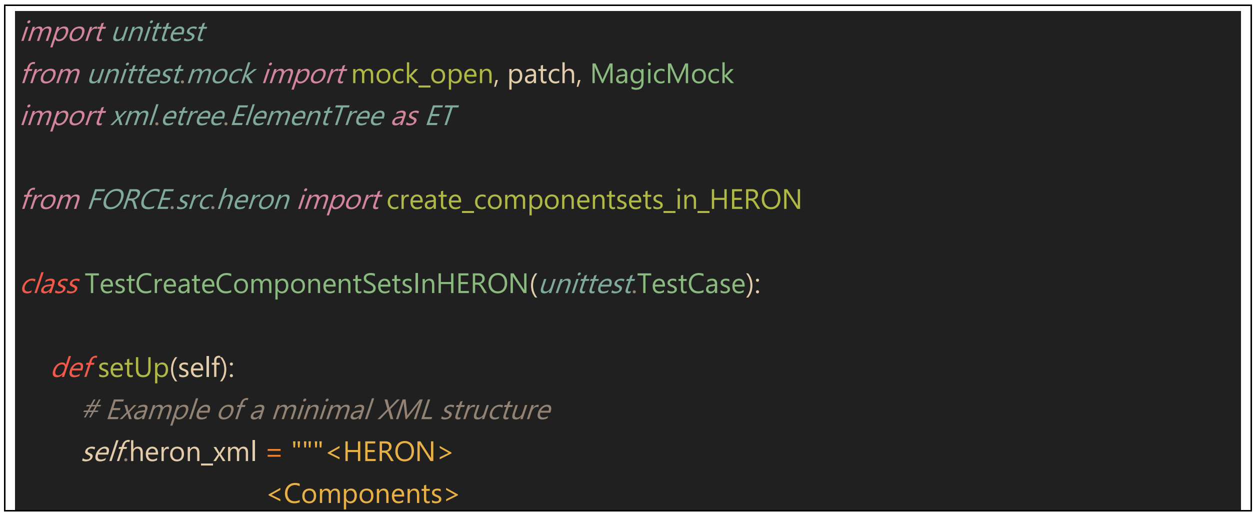
As discussed in previous sections, integration testing involves testing the interactions between integrated units or components of a software application to ensure that they work together as expected. This type of testing focuses on the interfaces and data flow between modules. So, with the implementation of both integration and unit tests, a full suite of testing, along with its infrastructure, has been merged into FORCE's main codebase. In the future, as code is developed and added to the FORCE repository, more unit tests will be written to ensure that new features are well tested at the most atomistic scale.

Table 2 summarizes the unit tests required in FORCE to reach 100% code coverage, and Figure 8 provides an example of what a unit-testing module looks like.

Table 2. A list of all unit tests required in FORCE to reach 100% code coverage.

File Name	Description	Testing Criteria
`test_heron.py`	Testing module looking at the conversion of data between all codes that are put into the HERON xml input format	- XML construction is correct - All data transformations are correct
`test_force.py`	Testing module looking at parsing info from JSON files and combining it into a	- Data object creation happens as expected

	data object that can create one holistic JSON file	- JSON data is transferred fully from one format to another
`test_apea.py`	Testing module looking at process of parsing JSON data and CSV data from APEA and outputting it as a JSON file	- Data is correctly read-in from CSV, TXT files - Data is correctly output to JSON format
`test_hysys.py`	Testing module looking at data conversions specific to HYSYS and output in JSON format	- Data is correctly read-in from XLSX - Data is correctly output to JSON format



```
import unittest
from unittest.mock import mock_open, patch, MagicMock
import xml.etree.ElementTree as ET

from FORCE.src.heron import create_componentsets_in_HERON

class TestCreateComponentSetsInHERON(unittest.TestCase):

    def setUp(self):
        # Example of a minimal XML structure
        self.heron_xml = """<HERON>
                        <Components>
```

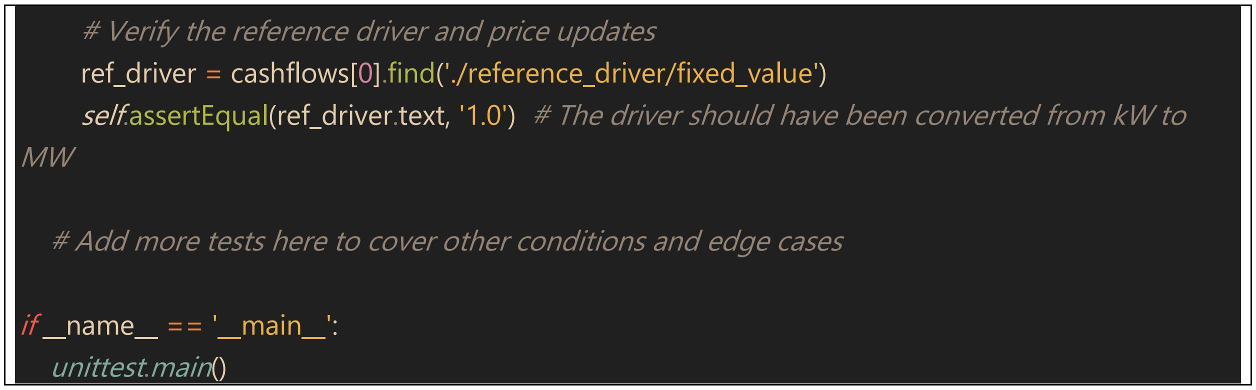
```
                                <Component name="ExistingComponent">
                                    <economics>
                                        <CashFlow name="ExistingComponent_capex">
                                            <fixed_value>100</fixed_value>
                                        </CashFlow>
                                    </economics>
                                </Component>
                            </Components>
                        </HERON>"""
    self.tree = ET.ElementTree(ET.fromstring(self.heron_xml))

@patch('xml.etree.ElementTree.parse')
@patch('os.listdir')
@patch('builtins.open',
       new_callable=mock_open,
       read_data="""{
                        "Component Set Name": "NewComponent",
                        "Reference Driver": 1000,
                        "Reference Driver Power Units": "kW",
                        "Reference Price (USD)": 2000,
                        "Scaling Factor": 0.5
                    }""")
def test_new_component_creation(self, mock_file, mock_listdir, mock_parse):
    # Setup the mock to return an XML tree
    mock_parse.return_value = self.tree
    mock_listdir.return_value = ['componentSet1.json']

    # Call the function
    result_tree = create_componentsets_in_HERON("/fake/folder", "/fake/heron_input.xml")

    # Verify the XML was updated correctly
    components = result_tree.findall('.//Component[@name="NewComponent"]')
    self.assertEqual(len(components), 1)
    economics = components[0].find('economics')
    self.assertIsNotNone(economics)

    # Verify the CashFlow node was created
    cashflows = economics.findall('CashFlow')
    self.assertEqual(len(cashflows), 1)
    self.assertEqual(cashflows[0].attrib['name'], 'NewComponent_capex')
```



```
        # Verify the reference driver and price updates
        ref_driver = cashflows[0].find('./reference_driver/fixed_value')
        self.assertEqual(ref_driver.text, '1.0')  # The driver should have been converted from kW to
MW

    # Add more tests here to cover other conditions and edge cases

if __name__ == '__main__':
    unittest.main()
```

Figure 8. Example of a unit test written for FORCE.

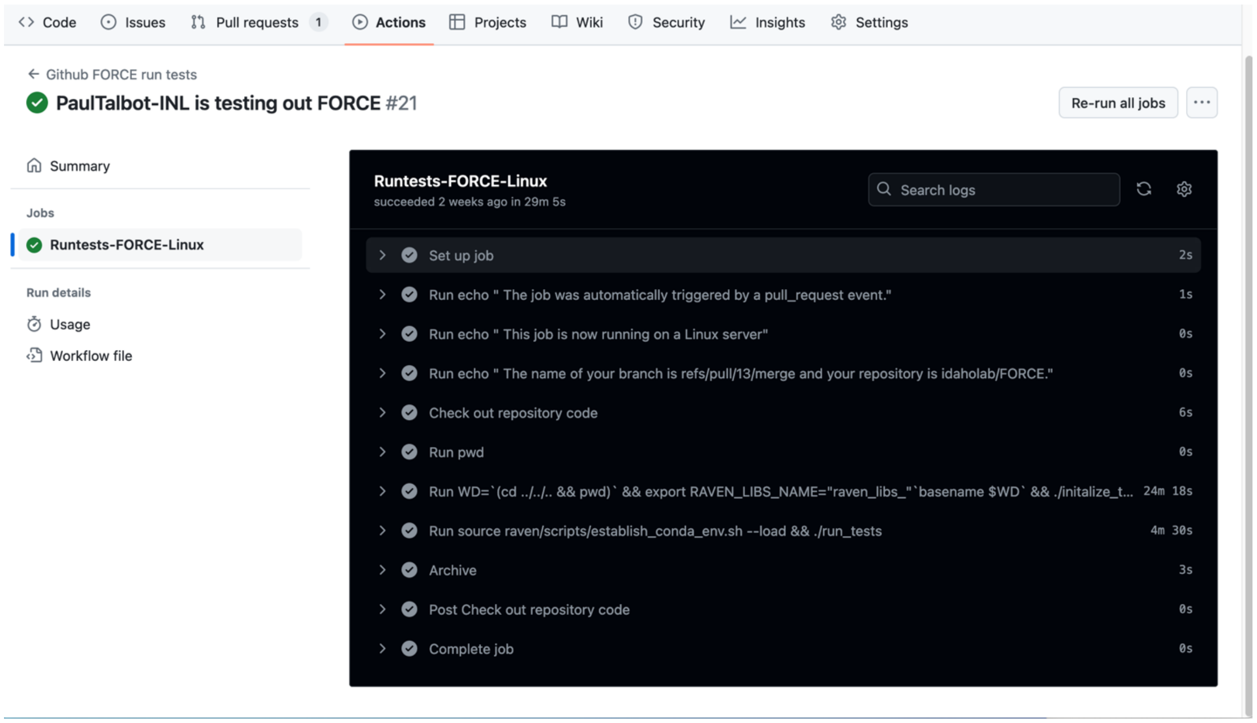
4. TESTING AUTOMATION (GITHUB ACTIONS)

GitHub actions for testing FORCE were added to automate the testing of FORCE.[a] GitHub actions allow various scripts to be run each time an event like a "pull request" or a "commit" occurs. These actions can be used for continuous integration. The GitHub action we created for this milestone allowed FORCE to automatically run the tests we added for this work each time a new pull request was created, and each time a push was made to the main FORCE repository. We created a yaml file in the github/workflows directory to specify the action. As part of the creation of this action, submodules for Risk Analysis Virtual Environment (RAVEN) and HERON were generated in the FORCE repository. These submodules specify the "git commit" that both RAVEN and HERON use as part of the initialization of the tests. Figure is a screenshot of the webpage after the action was run.

These are the steps that are used to run the tests as part of the GitHub action:

1. Check out the FORCE repository
2. Initialize the RAVEN and HERON submodules
3. Initialize the Tool for Economic Analysis (TEAL) submodule inside of RAVEN
4. Install HERON and TEAL as plug-ins in RAVEN
5. Create the RAVEN environment
6. Build RAVEN
7. Install openpyxl, which FORCE tests need
8. Run the run_tests scripts in FORCE
9. Gather up the test results in case they are needed for debugging.

[a] Pull request: https://github.com/idaholab/FORCE/pull/10



PaulTalbot-INL is testing out FORCE #21

Runtests-FORCE-Linux
succeeded 2 weeks ago in 29m 5s

```
Set up job                                                                                                   2s
Run echo " The job was automatically triggered by a pull_request event."                                     1s
Run echo " This job is now running on a Linux server"                                                        0s
Run echo " The name of your branch is refs/pull/13/merge and your repository is idaholab/FORCE."             0s
Check out repository code                                                                                    6s
Run pwd                                                                                                      0s
Run WD=`(cd ../../.. && pwd)` && export RAVEN_LIBS_NAME="raven_libs_"`basename $WD` && ./initalize_t...  24m 18s
Run source raven/scripts/establish_conda_env.sh --load && ./run_tests                                    4m 30s
Archive                                                                                                      3s
Post Check out repository code                                                                               0s
Complete job                                                                                                 0s
```

Figure 9. Screenshot of FORCE GitHub action result.

REFERENCES

1. American Society of Mechanical Engineers. 1982. *ANSI/ASME NQA 1c-1982 Addenda to ANSI/ASME NQA 1-1979 Quality Assurance Program Requirements for Nuclear Power Plants*. New York: The Society.
2. Idaho National Laboratory (2024a) The Framework for Optimization of ResourCes and Economics (FORCE). https://github.com/idaholab/FORCE
3. Idaho National Laboratory (2024b) HYBRID. https://github.com/idaholab/HYBRID
4. Idaho National Laboratory (2024c) Holistic Energy Resource Optimization Network (HERON). https://github.com/idaholab/HERON
5. Idaho National Laboratory (2024d) Optimization of Real-time Capacity Allocation (ORCA). https://github.com/idaholab/ORCA
6. GitHub. n.d. "GitHub Actions Documentation." Accessed May 20, 2024. https://docs.github.com/en/actions.
7. Hanna, Botros N., and Paul W. Talbot. *FORCE Development Status Update: Vertical Integration*. Report No. INL/RPT- 23-73267-Rev000. Idaho Falls: Idaho National Laboratory, 2023.
8. Python Software Foundation. Unittest—Unit Testing Framework. In Python 3.10.6 Documentation. 2024. Retrieved from https://docs.python.org/3/library/unittest.html.
9. Saeed, Rami M., Botros N. Hanna, and Paul W. Talbot. *FORCE Development Status Update: Vertical Integration and Benchmarking of System Dynamics*. Report No. INL/RPT-22-67745-Rev000. Idaho Falls: Idaho National Laboratory, 2022a.
10. Saeed, Rami M., Amey Shigrekar, Daniel M. Mikkelson, Aidan C. George Rigby, Courtney M. Yang Hui Otani, Marisol Garrouste, Konor L. Frick, and Shannon M. Bragg-Sitton. *Multilevel Analysis, Design, and Modeling of Coupling Advanced Nuclear Reactors and Thermal Energy Storage in an Integrated Energy System*. Report No. INL/RPT-22-69214-REV000. Idaho Falls: Idaho National Laboratory, 2022b. https://doi.org/10.2172/1890160.
11. Wendt, Daniel S., Marisol Garrouste, William D. Jenson, Qian Zhang, Mohammad S. Roni, Frederick C. Joseck, Tanveer H. Bhuiyan, and Richard D. Boardman. *Production of Fischer-Tropsch Synfuels at Nuclear Plants*. Report No. INL/RPT-22-69047-REV000. Idaho Falls: Idaho National Laboratory, 2022. https://doi.org/10.2172/1892316.