SD125713

Automated Testing Strategies for Add-ins

Jesse Rosalia
Founder, Bommer
jesse@bommer.io

Learning Objectives

- Set up our project for testing, and write unit tests for code that uses Fusion 360 objects.
- Write automated integration tests for add-in commands and user interfaces, and run them in Fusion 360.

Descriptions

We all know that testing your code is a critical step in deploying software, but manually following a script and pressing buttons on each screen is time consuming and error prone. In this class, we'll cover various testing strategies for Fusion 360 add-ins written in python, including structuring your add-in for unit testing, and automating integration testing. The goal of these is to ensure that your add-in is easily tested and verified, which will drive down bugs and drive up user satisfaction. This class is geared towards advanced programmers and add-in developers, and will cover Fusion 360 command and command definition APIs, events, and additional software used at Bommer to automate our testing. (Joint AU/Forge DevCon class).

Code Used During Presentation

Sample add-in project with test project: https://github.com/theJenix/au2017
Sodium testing framework: https://github.com/bommerio/sodium
Mock support: https://github.com/bommerio/f360mock

Refer to README.me files in code repositories listed above for detailed usage information!

Key Points

- Project setup is important! I recommend a project structure that places your test folder along side your add-in folder, e.g.
  - au2017
  - au2017/AU2017AddIn
    - *au2017/AU2017AddIn/AU2017AddInAddIn.py*
    - *au2017/AU2017AddIn/AU2017AddIn.manifest*
    - *au2017/AU2017AddIn/resources*
    - au2017/test
    - *au2017/test/__init__.py:*
      - `import sys`

```
import os
sys.path.append(os.path.abspath('AU2017AddIn'))
```

- Construct your tests using Python's built in unittest framework:
  - ```
    class ExampleTestCase(unittest.TestCase):
        def setUp(self):
            self.expected_values = …
            self.undertest = Example()

        def test_method(self):
            actual = self.undertest.method()
            self.assertEqual(self.expected, actual)

        def tearDown(self):
            pass
    ```

- Use f360mock to mock Autodesk Fusion 360 objects, for testing code with tight coupling to Fusion 360 API
  - Preferable over simple mock object or "duck typed" object
    - Strict mocking in dynamic language gives you assurance that you are calling the right things.
    - Tight coupling between mocks and adsk.core and adsk.fusion means you will know when it changes.
- Use Sodium to automate UI tests by defining unittest-like test cases that can operate on running commands:
  - ```
    class ExampleCommandTestCase(sodium.CommandTestCase):

        def __init__(self):
            super().__init__('Example')
            self._close_cmddef = get_command_def('SelectCommand')

        def setUp(self):
            cmdid = …
            app = adsk.core.Application.get()
            return app.userInterface.commandDefinitions.itemById(cmdid)

        def testCommand(self, command, inputs):

            # modify the inputs

            success = command.doExecute(False)
            assert success

            # test that command was executed

        def tearDown(self):
            self._close_cmddef.execute()

    runner = sodium.SodiumTestRunner()
    def run_tests():
        global runner
        runner.add(ExampleCommandTestCase())
        runner.run()
    ```

- Three step process to writing tests:
    - Identify the IDs of the command inputs to change
    - Identify how to verify the effects of the command
    - Write the testcase class.
- Use Sodium Inspector to inspect running commands, to find testable IDs.
- Sodium is under active development, feedback and contributions are welcome!

About the Speaker

Jesse Rosalia is the founder of Bommer, and the developer of the Bommer plug-in BOM management system. He has over 17 years experience in software development and architecture, including 2 years as the CTO of a small robotics startup (where he also dabbled in mechanical and electrical engineering), and is a mentor at the Highway1 hardware accelerator.

Jesse is keenly interested in the intersection of hardware and software, particularly the use of software tools to aid in the hardware development process, and has a passion for building practical, useful, usable tools that delight and improve the lives of users everywhere. Jesse is a proud to be an alumnus of Georgia Tech, and lives in San Francisco with his wife, Rachel, and 2 cats: Alan Puring and Ada Lovemice.

He can be reached at jesse@bommer.io for questions or feedback.