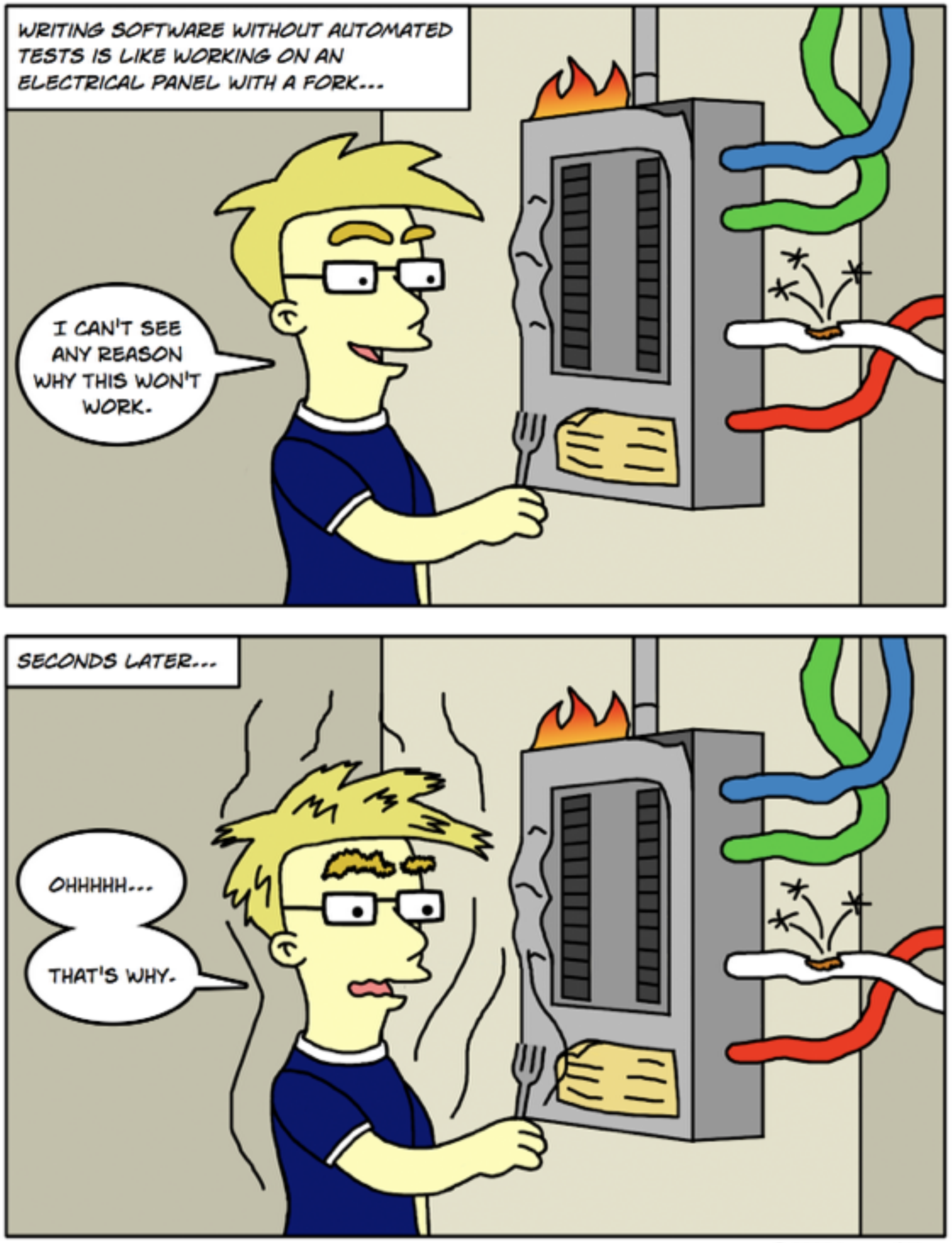
Unit testing in RMG

Belinda Slakman
July 25, 2014


WRITING SOFTWARE WITHOUT AUTOMATED TESTS IS LIKE WORKING ON AN ELECTRICAL PANEL WITH A FORK...
I CAN'T SEE ANY REASON WHY THIS WON'T WORK.
SECONDS LATER...
OHHHHH...
THAT'S WHY.

Today we'll discuss…

- Basics of unit testing
- Best practices
- Examples of some unit tests in RMG-Py
- An analysis of our code coverage

Key advantages of Python unit testing are (from quintagroup.com):

- Detecting problems early - Unit tests disclose problems early into the development.

- Mitigating change - Allows the developer to refactor the source code during the testing stage and later on, while still making sure the module works as expected.

- Simplifying integration - By testing the separate components of an application first and then testing them altogether, integration testing becomes much easier.

- Source of Documentation

Basic concepts of unit testing

- A `TestCase` is a groups of tests, while a `TestSuite` is a group of `TestCases` (or `TestSuites`)
- Nomenclature: *failures* are unexpected results, while everything else is an *error*
- In RMG-Py, live alongside the file it tests

Types of assertions

- `assertEqual()`
  - `assertAlmostEqual()`
- `assertRaises()`
- `assertTrue()`
- `assertIsInstance()`
- You can add the ‘msg’ option to make your output more meaningful (loops)

Best practices

- Use `setUp()` and `tearDown()`:
  - `setUp()`: runs before every test in a `TestCase;` can initialize variables or objects that are used in several tests
  - `tearDown()`: runs after every test, independently of whether it passed
- Regardless of us using ‘make test’ with nose, learn how to run your tests individually, and do it as you make commits to your specific code
- Write tests as you’re coding (TDD)
- Include general cases and “edge” cases
- Don’t delete any tests!
- Good unit tests can act as documentation
- Use one assertion per test case!
- Be careful about testing a function that relies on another function you are testing
  - Helps ensure that your code is modular and decoupled!

Ex: the second function fails if the first is faulty… a pain for testing

```
def is_prime(number):
    """Return True if *number* is prime."""
    for element in range(number):
        if number % element == 0:
            return False
    return True

def print_next_prime(number):
    """Print the closest prime number larger than *number*."""
    index = number
    while True:
        index += 1
        if is_prime(index):
            print(index)
```

Examples

```
class TestSoluteDatabase(TestCase):

    # good practice to use this!
    def setUp(self):
        self.database = SolvationDatabase()
        self.database.load(os.path.join(settings['database.directory'], ‘solvation’))

    # okay example
    def testDiffusivity(self):
        "Test that for a given solvent viscosity and temperature we can calculate a solute's
        diffusivity"
        species = Species(molecule=[Molecule(SMILES='COC=O')])
        soluteData = self.database.getSoluteData(species)
        T = 298
        solventViscosity = 0.001
        D = soluteData.getStokesDiffusivity(T, solventViscosity)
        self.assertAlmostEqual((D*1E12), 0.00000979)

    # bad example
    def testSolventLibrary(self):
        "Test we can obtain solvent parameters from a library"
        solventData = self.database.getSolventData('water')
        self.assertTrue(solventData is not None)
        self.assertEqual(solventData.s_h, 2.836)
        self.assertRaises(DatabaseError, self.database.getSolventData, 'orange_juice')
```

Running tests

- Simple:

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

- More control:

```
suite = unittest.TestLoader().
          loadTestsFromTestCase(TestSequenceFunction)
unittest.TextTestRunner(verbosity=2).run(suite)
```

- We use nose.py to run all of our tests at once

Nose

- Runs when we ‘make test’
- Collects and runs TestCases
- Plugins for collecting information (both built-in or user-written)
    - Coverage (we’ll discuss ours later)
    - Error handling
    - Printing output

Coverage in RMG-Py

- Learn about coverage: http://nedbatchelder.com/code/coverage/
- Measures % lines of executable code that have been executed
- It’s stand-alone, but we use the nose plug-in
- Info stored in testing/coverage

Our results

Helpful resources

- Python documentation
- http://jeffknupp.com/blog/2013/12/09/improve-your-python-understanding-unit-testing/
- http://pymotw.com/2/unittest/ “Python module of the week”

Suggestions

- Practice test driven development!
- Improve current unit tests for “your” code, and assign people to write tests for other stuff that’s lacking
- …..?