Accommodating Test Dependence In Regression Testing Algorithms

Jonathan Xue

Mentors: Wing Lam, Reed Oei

Background

- Software tests enable developers to quickly identify errors and bugs [1]
- Regression Testing
  - The process of testing existing software functionality after the introduction of modifications within the code
  - Three main techniques:
    - Test Prioritization: Runs the tests in an execution order such that tests which are likely to fail are ran first
    - Test Selection: Runs a subset of tests such that only the tests affected by modified elements are ran
    - Test Parallelization: Runs tests simultaneously across multiple machines/CPUs
- Dependent Tests
  - Tests which yield different test results depending on the order of the test suite (example 1)

```java
public class DependentTestsExample() {
    // Variable
    public static int x = 0;

    @Test
    public void testAddition() {
        // Increments X
        x++;
        assertEquals(x, 1);
    }

    @Test
    public void testSubtraction() {
        // Decrements x
        x--;
        assertEquals(x, 0);
    }
}
```

Example 1. The addition and subtraction functions are both valid yet depending upon the order in which the test suite is run, a false negative may occur. *testAddition* → *testSubtraction* results in two successful tests, while *testSubtraction* → *testAddition* results in two failing tests

My Work

- Write two separate Maven Plugins to help automate the process of accommodating dependent tests within testing algorithms
  - Plugin 1 (runs on version w/o modifications):
    - Instruments source and test classes to gather time and coverage information of each test for regression testing techniques
    - Precompute test dependencies
  - Plugin 2 (runs on version w/ modifications):
    - Accommodate test dependencies on new versions using precomputed dependencies from Plugin 1. Test outputs should now ideally no longer contain false positives or negatives due to order-dependent tests

Process Without My Work

Step 1: Download the necessary repositories
Step 2: Setup the necessary variables (e.g. versions, paths, dates)
Step 4: Precompute test dependencies
Step 5: Move necessary files to their designated locations
Step 6: Setup necessary variables
Step 7: Run regression testing algorithms while accommodating test dependence

Process With My Work

Step 1: Insert two blocks of XML code into the pom.xml file of both versions
Step 2: Version w/o modifications
mvn testrunner:<plugin1>
Step 3: Version w/ modifications
mvn testrunner:<plugin2>

Benefits

- Plugins work on any Maven repository
- Plugins reduce the process to three simple steps with trivial manual effort
- Use of our work is 7.1% faster at producing reliable outcomes than regression testing algorithms that assume test independence [2]

Importance

- Helps developers accommodate dependent tests so that they are not blocked by false positives/ negatives and can focus on more pressing issues (e.g., shipping new features)
- Microsoft estimated that for complicated systems like Windows, the cost of test result inspections (i.e. verifying if test failures are due to dependent tests) can cost $2 million a year [1]

References

1. Kim Herzig, Michaela Greiler, Jacek Czerwonka, and Brendan Murphy. 2015. The art of testing less without sacrificing quality. In *ICSE'15, Proceedings of the 37th International Conference on Software Engineering*. Florence, Italy, 483–493.
2. Sai Zhang, Darioush Jalali, Jochen Wuttke, Kıvanc Muslu, Wing Lam, Michael D. Ernst, and David Notkin. 2014. Empirically revisiting the test independence assumption. In *ISSTA 2014, Proceedings of the 2014 International Symposium on Software Testing and Analysis*. San Jose, CA, USA, 385–396.

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

400 Engineering Hall, MC 268
1308 West Green Street
Urbana, IL 61801

PURE
Promoting Undergraduate Research in Engineering