4-30-2021

Automated Code Testing

Sam Lukins

Austin Seidler

Automated Code Testing

by

Austen Seidler and Sam Lukins

An Honors Capstone

submitted in partial fulfillment of the requirements

for the Honors Diploma

to

The Honors College

of

The University of Alabama in Huntsville

April 27, 2021

Honors Capstone Director: R. Kevin Preston

Computer Science Lecturer

Sam Lukins 2021-04-29

Student (signature) Date

R. Kevin Preston 30-Apr-2021

Director (signature) Date

[illegible] Date: 2021.04.30 11:13:44 -05'00'

Department Chair

Department Chair (signature) Date

William Wilkerson Digitally signed by William Wilkerson
Date: 2021.04.30 16:08:57 -05'00'

Honors College Dean (signature) Date

HONORS COLLEGE
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Honors College
Frank Franz Hall
+1 (256) 824-6450 (voice)
+1 (256) 824-7339 (fax)
honors@uah.edu

Honors Thesis Copyright Permission

This form must be signed by the student and submitted as a bound part of the thesis.

Sam Lukins

Student Name (printed)

Sam Lukins

Student Signature

2021-04-29

Date

Table of Contents

Abstract

Software testing is the process of verifying that a software application meets expected requirements and is free of defects. When software tools are used to test a software application and verify the results without manual human intervention, it is known as automated testing. To better understand the field of software testing, we wrote a series of automated tests for our group's senior design project, following recommended guidelines for software test development. We then observed the results of our testing and the perceived benefits of using automated tests.

Introduction

Software testing is a vital yet time-consuming aspect of software development. In fact, according to one estimated, testing can often consume around 40~50% of software development efforts [1]. However, despite the importance of testing, software testing skills are often in short supply, especially among recent graduates, who often undervalue its role in software development and lack sufficient hands-on experience. This is especially true with regards to automated testing, as students often lack knowledge on how to write effective and complete automated tests. One 2016 study on perceptions of automated testing found that 55% of participants, all of whom were employed software developers, were not satisfied with the testing skills that new hires possessed [2]. This study found that these new hires generally lacked both knowledge of automated testing and hands-on experience with writing and using automated tests [2].

In this paper, we will be discussing our own work with automated testing for our senior design project and the recommended guidelines picked up from research and our own experience, as well as reporting the perceived benefits of writing automated tests for our project. Our hope is that this project will serve both as an example of how automated testing can be included alongside other projects undertaken by those new to the field of software development and as a guide detailing what practices to follow when designing automated tests.

Testing Environment

The Project

First, it is important to understand the project that was being tested. Our group's assignment was to develop a program that can simulate a robotic vacuum cleaner in order to determine the effectiveness of various pathing algorithms that can be used by the vacuum. To accomplish this, the program utilized three major objects: a house object, a vacuum object, and a simulation runner, all of which required testing.

The first of these objects, the house, consisted of rooms, doors, furniture, and a floor covering. These parts could be edited using a graphical user interface (GUI) to allow for testing the vacuum's algorithms within any number of differing floor plans that could be saved to and loaded from files. Functions used by the house and related classes to accomplish these tasks included methods for adding rooms and furniture, methods for removing rooms and furniture, a method for adding a door to a room's wall, a method for setting the floor covering, and methods for serializing and deserializing a house to a given file.

The second object, the vacuum, had several different attributes that needed to be kept track of, including both fixed and variable attributes. Fixed attributes included size, vacuum width, and whisker width. Variable attributes included vacuum efficiency, whisker efficiency, speed, battery life, and pathing algorithm.

The third object, the simulation runner, was the most complex. The simulation would allow for the user to observe a vacuum running in real time, with the ability to run the simulation at x1, x5, and x50 speeds. As the vacuum moved, color coding was used to display which areas were cleaned. Once the simulation finished running, the results would be saved to a file, which could then be loaded back in to compare them to the results of previous runs. This object had many functions requiring extensive testing, most notably when it came to collision detection.

Software Tools

The robot vacuum simulator was written in Java using NetBeans and IntelliJ IDEA, while the JUnit testing framework was used to write automated tests. Many tests also used the Mockito framework alongside JUnit.

Test Design

Once work on the program was underway, automated tests were needed to ensure that everything functioned as intended. These tests were written alongside the code they were intended to test so that we could verify as soon as possible that everything was working correctly. While writing tests, certain testing practices were followed in order to ensure that tests were of high quality.

Short, Meaningful Tests

Automated tests should be short, having only a single main focus per test. For example, consider the following test, which was written to test removing a room from a house.

```
@Test
public void testRemoveRoom() {
    House h1 = new House(20, 25, FlooringType.HARD);
    h1.addRoom(5, 5, 4, 4);
    assertNotNull(h1.getRoom(new Position(5, 5)));
    h1.removeRoom(new Position(5, 5));
    assertNull(h1.getRoom(new Position(5, 5)));
}
```

This test essentially performs a single main task: it asserts that a room that was previously determined to exist no longer does once the removeRoom() function is called. During the course of our project, we found that writing tests in this way resulted in simple, easily comprehensible tests.

Mock Objects Not Being Tested

Objects that are not the focus of a test should be mocked using a framework such as Mockito, as mocking an object creates a more isolated object without dependencies that could needlessly clutter tests. It is for this reason that mocking is commonly used by developers when faced with classes too complex to be instantiated in the test code [3]. The following function is an example of a method that test cases for this program used to create mock versions of collision circles:

```
private CollisionTestData generateMockCollisionCircleTestData(double x,
double y, double radius) {
    CollisionCircle circleMock = Mockito.mock(CollisionCircle.class);
    when(circleMock.getRadius()).thenReturn(radius);
    when(circleMock.getShape()).thenReturn(Shape.CIRCLE);
    Position pos = new Position(x, y);
    CollisionTestData ctdMock = Mockito.mock(CollisionTestData.class);
    when(ctdMock.getcShape()).thenReturn(circleMock);
    when(ctdMock.getPos()).thenReturn(pos);
    return ctdMock;
}
```

A mocked collision circle created using this function contains all of the information a test might need from a collision circle without the same level of complexity that creating a real collision circle would involve. We found that the introduction of functions such as this one made writing tests much easier.

Test Edge Cases

Edge cases are scenarios that occur during extreme situations that are unlikely to occur in normal operation. Despite being uncommon, testing edge cases is an important part of software testing [3], ensuring completeness of test coverage. For example, when testing static collisions between simulated objects, tests were written not just for basic collisions but also for unlikely scenarios such as an object colliding with the corner of a wall and an object colliding when entirely inside another object.

Make Tests Independent

Automated tests should be entirely independent from each other. A test should not depend on another test in order to function, and the performance of a test should not affect any other test’s performance, as making tests that are dependent on each other can create a single point of failure that causes multiple tests to fail. For our project, we ensured that every test could be run independently and found that it also led to cleaner tests that were easier to understand.

Results

In total, 34 automated tests were created for the robot vacuum simulator using these guidelines over the course of the semester. The part of the program that required the most testing was collision detection, as there were many possibilities that could occur during execution that could cause the program to fail if just one did not work properly.

In addition to writing tests, a PowerPoint presentation detailing these testing guidelines was created to be used to educate students on how to test code more effectively. The presentation included definitions, good practices, things to avoid, and examples of how tests are formatted.

Perceived Benefits of Automated Testing

Because of these tests, we were able to verify the functionality of many important methods and discover errors that may have otherwise eluded us. Additionally, the automated nature of the tests meant that they had a couple of extra benefits. The first of these benefits was that the tests could be run much more quickly and accurately than tests performed manually. The other main benefit we observed was that the automated tests were easily repeatable, allowing for tests to be re-run whenever changes to the program code were made that might affect the results of one or more tests. Of course, we are not alone in reporting these benefits. Previously performed studies have reported similar findings, including a 2012 survey where 72% of respondents agreed that the ability to re-run automated tests saves time and cost [4]. Of course, automated tests still have their limitations. For example, there were some errors with the robot vacuum simulator that went undetected by the automated tests due to the errors being more complex than the incorrect operation of any single function or simple task. These errors were instead discovered during execution of the program.

Conclusion

Automated testing is an incredibly deep and complex field with all sorts of practices, definitions, and guidelines to study and analyze, many of which are beyond the scope of this project. Some potential continuations of this project include researching the differences between the tests performed here and other testing techniques such as regression testing, comparing the effectiveness of automated testing on differing software applications, or performing a more in-depth study on the design of testing frameworks like JUnit. Regardless of what future research into automated research focuses on, the work presented here should serve as a useful look at good practices for automated testing and a potential guide for those looking to enter the field.

Reference List

[1] G. Suganya and S. Neduncheliyan, "A study of Object Oriented testing techniques: Survey and challenges," 2010 International Conference on Innovative Computing Technologies (ICICT), Karur, India, 2010, pp. 1-5, doi: 10.1109/ICINNOVCT.2010.5440079.

[2] R. Pham, S. Kiesling, L. Singer, K. Schneider, “Onboarding inexperienced developers: struggles and perceptions regarding automated testing”, in *Software Quality Journal*, vol. 25, no. 4, New York: Springer US, 2017, pp. 1239-1268, doi: 10.1007/s11219-016-9333-7

[3] M. Aniche, C. Treude, A. Zaidman, “How developers engineer test cases: an observational study”, 2021

[4] Dudekula Mohammad Rafi, Katam Reddy Kiran Moses, K. Petersen and M. V. Mäntylä, "Benefits and limitations of automated software testing: Systematic literature review and practitioner survey," 2012 7th International Workshop on Automation of Software Test (AST), 2012, pp. 36-42, doi: 10.1109/IWAST.2012.6228988.