EECS4313 week 2

Jerry Wu

2024-01-22

Contents

Lecture 1

Test automation

Why automated testing?

- Manual testing is time consuming and not economical
- Automation is **unattended** and **can happen overnight**; thus not requiring human intervention
- Automation increases the speed of test execution
- Manual testing is error prone due to its recurring nature

Executing automated tests

- Test automation is the use of software separate from the software being tested to control the execution of tests
  - Generating test inputs and expected results
  - Running test suites without manual intervention
  - Evaluating pass/no pass
- Testing must be automated to be **effective and repeatable**

What can be automated?

Each one of these steps in software development can be automated, and each step down is easier to automate than the last. Of course, there has to be some human intervention to formulate the requirements for automation at the start.

- **Analyze** - intellectual (performed once)
- **Design** - mostly intellectual
- **Construct** - clerical and intellectual
- **Testing** - mostly clerical
- **Deploy** - clerical (repeated many times)

The more clerical something is, the more worth automating it is.

Record & playback

- Usually for websites, mobile, UIs, etc.
- Test scripts are recorded on the initial version of the application
- The same scripts are executed on the next version
- The scripts need **some modification** (quite costly at high number of modification) for the changes happening on the application during every version of the application
- The test scripts repository keeps growing as the application goes through changes. This makes this kind of test suite very hard to maintain

Procedure

- Automation tool generates scripts by recording user actions
- The generated scripts can be played back to reproduce the exact user actions

Advantages

- Less effort for automation
- Quick returns
- Does not require expertise on tools

Limitations

- High dependency on the GUI of AUT (application under test)
- Difficult to maintain the scripts

Problems

- Very fragile
  - A single change in UI can cause the whole system to break
- Hard to maintain
  - An abundance of test scripts causes the test suite to be hard to maintain
- No modularity or reuse
  - System must be ready before automation can start

Testing terminology contd'

> *"You don't want to make assumptions. That is not unit testing" - H.V. Pham 2024*

- A **unit test** is a test of a single class/method
- A **test case** tests the respose of a single method to a particular set of inputs
- A **test suite** is a collection of test cases
- A **test runner** is software that runs tests and reports results
- A **flaky test** is a test case where it is sometimes triggered and sometimes not depending on specific conditions, seemingly randomly.

Std. structure of a JUnit test class

- Test a class called `Fraction`
- Create a test class called `FractionTest`

```
1  import org.junit.*;
2  import static org.junit.Assert.*;
3
4  public class FractionTest{
5      @Test
6      public void addTest(){...}
7
8      @Test
9      public void testToString(){...}
10
11     //useful if we have a counter during the tests to set in SetUp() and
           reset in TearDown()
12     @Before
13     public void SetUp(){...}
14
15     @After
16     public void TearDown(){...}
17 }
```

More fixtures for test classes

- There is also a `@BeforeClass` annotation that will execute once before **all** test cases
- Similarly, there is also an `@AfterClass` annotation that executes a method after every test case
- Usually done for expensive allocations for resources like connecting/disconnecting a database.