Test Automation

CS 3250
Software Testing

[Ammann and Offutt, “Introduction to Software Testing,” Ch. 3]
Tasks in Testing Process

- **Design tests**
- **Enter inputs, run program, record results**
  - Enter Test values (inputs)
  - Program
  - Actual results vs Expected results Comparison
  - Execute tests against the program under test (PUT) or sometimes referred to as a software under test (SUT)
- **Analyze results**
- **Revenue task**
- **Excise task**
Manual Testing

Imagine you are testing a web program that takes a drink order.

Load the web program
Enter all inputs
Click “Place order” button
Record the test result
Proceed to the next test

Awesome Shop!

Name:

Email:

Phone:

What would you like to drink?

Hot or cold?
- Hot
- Cold

Send me a text message when my order is ready

Place order
Manual Testing

Benefits

• Simple and straightforward
• No up-front cost
• Easy to set up
• No additional software to learn, purchase, write
• Flexible
• More likely to test things that users care about
• Possible to catch issues that automated testing does not

Drawbacks

• (Extremely) boring
• Often not repeatable
• Some tasks are difficult or impossible to test manually
• Human errors
• Extremely time- and resource-intensive
• Limit to black-box and grey-box testing
Software Testing Activities

1. Test Design

2. Test Automation

3. Test Execution

4. Test Evaluation

Human-based or Criteria-based

Test Engineer

Test Manager

test automate

Test Engineer

execute

Computer

Output

Human-based or Criteria-based

Test Engineer

design

Test Engineer

analyze

[AO, p.22]
Automated Testing

Benefits

• Reduce human errors
• Fast execution
• Easy to execute once the system is set up
• Easily repeatable
• Simple to analyze the process
• Less resource-intensive
• Cover aspects that manual testing is impossible
• Scalable

Drawbacks

• Setup time up-front
• May miss some user-facing defects
• Require training – programming languages, testing frameworks, tools
**Test Automation**

- Use of software to control the **execution** of tests
- **Comparison** of actual outcomes to predicted outputs
- **Setting up** of test preconditions, and other test **control** and test **reporting** functions

![Test Automation Diagram]

Executable Tests
- Precondition
- Test control
- Test inputs
- Expected results

Test values (inputs) → Program → Actual results vs Expected results

Comparison

Report
Benefits of Test Automation

Benefits of test automation

- Reduce excise tasks
- Free up time to focus on revenue tasks
- Reduce cost
- Reduces human error
- Reduce variance in test quality from different individuals
- Significantly reduce the cost of regression testing

Automate as much as possible
What Makes Test Automation Hard

Testability

• “The degree to which a system or component facilitates that establishment of test criteria and the performance of tests to determine whether those criteria have been met”

How hard it is to detect faults in the software

• Two practical problems:
  • **Controllability**: how to provide the test values to the software
  • **Observability**: how to observe the results of test execution
Aspects that Impact Testability

Controllability

• “How easy it is to provide a program with the needed inputs, in terms of values, operations, and behaviors”

• Example of software that often have low controllability:
  • Embedded software often gets its inputs from hardware sensors
    • Difficult to control
    • Some inputs may be difficult, dangerous, or impossible to supply
  • Component-based software
  • Distributed software
  • Web applications
  • Mobile applications
Aspects that Impact Testability

Observability

• “How easy it is to observe the behavior of a program in terms of its outputs, effects on the environment and other hardware and software components”

• Example of software that often have low observability:
  • Embedded software often does not produce output for human consumption
  • Component-based software
  • Distributed software
  • Web applications
  • Mobile applications
Dealing with Observability and Controllability

- Simulation – bypass the hardware or software components that interfere with testing

[Devi et al., 2019]
Importance of Testability

- Testability is crucial to test automation
- Test scripts need to control the execution of the component under test
- Test scripts need to observe the results of the test
Components of a Test Case

Test case
A multipart artifact with a definite structure
A test case is composed of test case values, expected results, prefix values, and postfix values necessary for a complete execution and evaluation of the software under test.

Test case values
The inputs needed to complete an execution of the software under test
Inputs can be input values or series of actions

Expected results
The result that will be produced by the test if the software behaves as expected

Test oracle (pass or fail)
Determine the testing quality
## Components of a Test Case

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prefix values</strong></td>
<td>Inputs needed to put the software into the appropriate state to receive the test case values.</td>
</tr>
<tr>
<td><strong>Postfix values</strong></td>
<td>Any inputs that need to be sent to the software after the test case values are sent.</td>
</tr>
<tr>
<td><strong>Verification values</strong></td>
<td>Values needed to see the results of the test case values.</td>
</tr>
<tr>
<td><strong>Exit values</strong></td>
<td>Values or commands needed to terminate the program or otherwise return it to a stable state.</td>
</tr>
</tbody>
</table>

Inputs can be input values or series of actions.

Additional components: test case ID, purpose of the test.
Putting Tests Together

• **Test case**
  - The test case values, prefix values, postfix values, and expected results necessary for a complete execution and evaluation of the software under test

• **Test set**
  - A set of test cases

• **Executable test script**
  - A test case that is prepared in a form to be executed automatically on the test software and produce a report
Test Case Execution

1. Setup/execute prefix values
2. Setup test case values
3. Execute SUT
4. Compare the results
   - Actual results vs Expected results
5. Setup/execute postfix values
## Example Test Cases: `numZero`

```java
public static int numZero(int[] x) {
    // Effects: if x == null, throw NullPointerException
    // else return the number of occurrences of 0 in x
    int count = 0;  // line1
    for (int i=1; i<x.length; i++) {  // line2
        if (x[i] == 0) {  // line3
            count++;  // line4
        }
    }
    return count;  // line5
}
```

<table>
<thead>
<tr>
<th>Test case ID</th>
<th>Test case values (x)</th>
<th>Expected results</th>
<th>Actual results</th>
<th>Failure?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>null</td>
<td>NPE</td>
<td>NPE</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>{}</td>
<td>0</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>{1, 2, 3, 4}</td>
<td>0</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>{1, 0, 2, 0}</td>
<td>2</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>{0, 1, 2, 0}</td>
<td>2</td>
<td>1</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Test Automation Framework

• A set of assumptions, concepts, and tools that support test automation

• Provides a standard design for test scripts and support for the test driver

• Test driver
  • Runs a test set by executing the software repeatedly on each test
  • Supply the “main” method to run the software if it is not standalone (i.e., a method, class, or other component)
  • Compares the results of execution with the expected results (from the test case) and report the results to the tester
Test Automation Frameworks (2)

• Most test automation frameworks support
  • Assertions to evaluate expected results
  • The ability to share common test data among tests
  • Test sets to easily organize and run tests
  • The ability to run tests from either a command line or a GUI

• Most test automation frameworks are designed for unit and integration testing, although some specifically support system testing, and some are built to support testing over the web

• Example test automation frameworks
  • JUnit, HttpUnit, HtmlUnit, JWebUnit, Selenium, unittest, Jasmine, PHPUnit, Robotium