CS @ ILLINOIS
COMPUTER SCIENCE

PURE
Promoting Undergraduate Research in Engineering

Parameterized Unit Testing in Java

Jinlin Xu Mentor: Wing Lam

Background Info

- **Conventional Unit Test (CUT)**

For CUT, developers write test cases one by one. When there are a lot of conditionals in the code under test, it is difficult to cover all lines of code.

A real-world CUT

```
public void CUT() {
  int a = 1;
  IntStack stack = new IntStack();
  stack.push(a);
  assertEquals(1, stack.count());
}
```

- **Parameterized Unit Test (PUT)**

For PUT, developers can use test generation tools like Pex to generate inputs for the parameters of the tests. Each set of inputs generated by Pex will cover as many uncovered lines of code as possible.

PUT corresponding to the CUT above

```
public void PUT(int[] numberArray){
  IntStack stack = new IntStack();
  for(int number: numberArray){
    stack.push(i); }
  assertEquals(numberArray.length, stack.count());
}
```

Motivation

My mentor's research is on open source C# projects. My aim is to study the usage of PUTs in Java open source projects and collect a variety of data on these projects. Improving and running the analysis tools on these projects will allow me to gather data to determine whether my mentor's findings are true for projects written in Java or not.

Research Plan

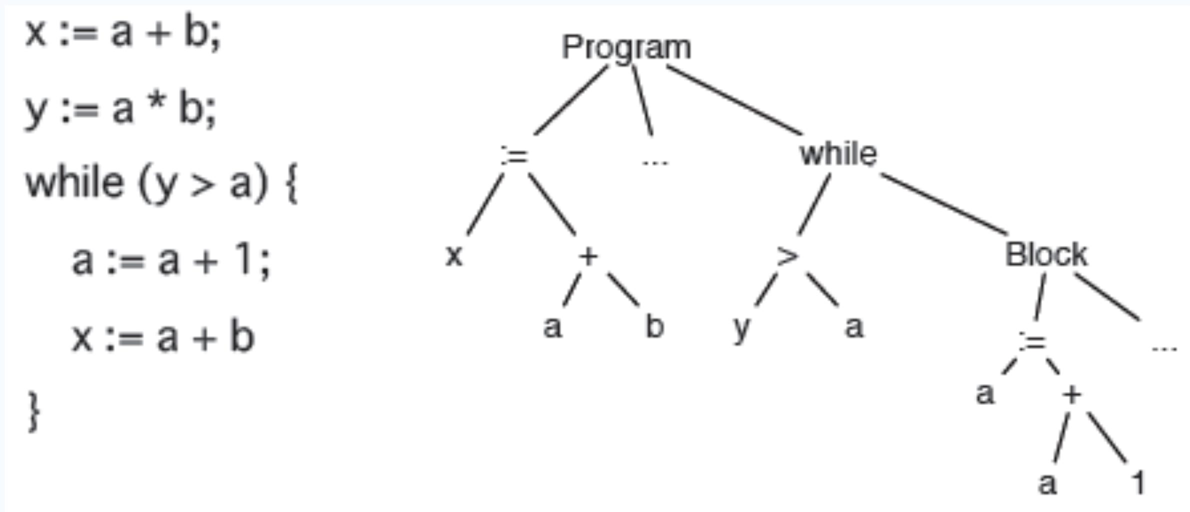
- Find and setup Java projects containing both CUTs and PUTs. The projects should be compilable and be compatible with our analysis tools.
- Improve our Java static analysis tools and use them to generate data for our Java projects. The tools should be able to collect information relevant to the findings found by our C# projects.

Progress to Date

- Study and write PUTs for Java methods.
- Setup the available Java open source projects built by different build systems (e.g., Maven, Ant).
- Edit the analysis tools to generate new data for the tables in my mentor's paper.
- The table on the right shows part of the data generated.

Data Structure in the Analysis Tool

The analysis tool uses an Abstract Syntax Tree (AST) parser to parse the source code and get data such as the number of methods invoked, number of parameters, and lines of code in a project. Here is an example of an AST.



Java Projects Analyzed

The analysis tool is ran on 11 Java open source projects. All of them contain both PUTs and CUTs.

Junit-Quickcheck
JUnit4
AssertJ
Java Git
Concourse
Atlassian Universal Plugin Manager
RoboBinding
LensKit
Java APNS
Izpack
Clojure Maven Plugin

Results

Running the tool on the projects, we get new data like lines of code, methods invoked by CUT, and their statement coverage. Below is the table for lines of code.

Project Name	#LOC		
	Source	PUT	CUT
Junit-Quickcheck	4871	482	825
JUnit4	25408	97	5372
AssertJ	109498	315	42230
Java Git	126401	690	35999
Concourse	47278	172	4775
Atlassian Universal Plugin Manager	24068	111	3486
RoboBinding	25193	145	1970
LensKit	36376	14	1707
Java APNS	5251	40	708
Izpack	76244	4	975
Clojure Maven Plugin	1414	4	13
Total	482002	2074	98060

Future Plan

- Continue to find and compile more Java open source projects to analyze with our tools.
- Improve the analysis tools to make it be able to collect all of the data needed for the tables in our paper. Currently, our tools is unable to count the assumptions, assertions, and attributes of a method.