Testing Techniques

Assignment 2

Sasja Gillissen, Martin Huijben, Martijn Terpstra

December 1, 2016

1 Automated Test Execution

In our previous assignment we developed the following use cases

Category	Input	Expected Output
Exiting	`exit`	Nothing
Function Assignment	`f(x):= x + 1 ; f(7)`	`8`
Builtin Functions (1)	`sqrt(ln(log(100)))`	`0.83`
Builtin Functions (2)	`floor(0.5) + ceil(0.5) + round(0.5)`	`2`
Variable Assignment	`k = 3 ; k`	`3`
Builtin Variables	`e * pi`	`8.5397342226735656`
Expression (1)	`(((0 + 1) * 2) - 3) / 4`	`-0.2500000000000000`
Expression (2)	`1 / 0`	Error
Equality (1)	`3 = 3`	`0`
Equality (2)	`2 = 1`	`1`
Equality (3)	`2 = 3`	`-1`

1.1 Describe, design, and develop an automated test execution environment for your SUT.

Our SUT is deterministic so for any given input we can expect the same output. As such we can simply test each use case by supplying input for each test case and seeing if the output matches.

For each use case we create 3 files with the post fixes "_input.txt" "_output.txt" and "_errors.txt". The first is the input we give the SUT, the other two being the expected outputs. These are split in two since UNIX shells output to both STDOUT (normal output) and STDERR(errors). Some of our use cases except the SUT to throw error output so this is included.

For the actual test we have written a simple bash script that.

- For each test pipes the input into our SUT and writes the output and errors to a file

- Compares the outputs with the expected output using the "diff" command
- If the outputs match simply echo "[OK]"
- If the outputs don't match, show a diff of the outputs outlining what was different.

The SUT passes all tests if all test cases return "[OK][OK]"

1.2 Implement the test cases that you developed for manual testing, as scripts, programs, text files, . . . , so that they can be executed with your test execution environment.

The full code, including the use cases can be found at `https://github.com/Witik/CommandlineCalculator`

1.3 Test your SUT with the automated test scripts and analyze the results.

This is the final output of our test script

```
exit:                 [OK][OK]
function_assignment:  [OK][OK]
builtin_function1:    [OK][OK]
builtin_function2:    [OK][OK]
variable_assignment:  [OK][OK]
builtin_variables:    [OK][OK]
expression1:          [OK][OK]
expression2:          [OK][OK]
equality1:            [OK][OK]
equality2:            [OK][OK]
equality3:            [OK][OK]
```

The result is unsurprising, each test is successful. Since we had previously defined all use cases in detail making correct tests turned out to be easy.

1.4 Evaluate your automated test execution environment.

Since we had previously made very concrete test cases and our program can be communicated with from the command line automation was easy. Once our script could test the first use case, the remaining ten use cases could be added to the test suite in a matter minutes.

The testing suite was kept simple, allowing to to be developed in a lunch break. It would most likely have taken longer to look for a tool, learn it and

set it up. The full testing environment could be setup in a fraction of the time it took us to get TorXakis working.

Furthermore because of its simplicity it was trivial to add use cases. Which means that if our SUT would later gain more features we could easily add more use cases to test for.

2 Model-Based Testing

2.1 Modeling Investigation

We wanted to model the use cases we made in the previous assignment. Here they are again:

Category	Input	Expected Output
Exiting	`exit`	Nothing
Function Assignment	`f(x):= x + 1 ; f(7)`	`8`
Builtin Functions (1)	`sqrt(ln(log(100)))`	`0.83`
Builtin Functions (2)	`floor(0.5) + ceil(0.5) + round(0.5)`	`2`
Variable Assignment	`k = 3 ; k`	`3`
Builtin Variables	`e * pi`	`8.5397342226735656`
Expression (1)	`(((0 + 1) * 2) - 3) / 4`	`-0.2500000000000000`
Expression (2)	`1 / 0`	Error
Equality (1)	`3 = 3`	`0`
Equality (2)	`2 = 1`	`1`
Equality (3)	`2 = 3`	`-1`

Here the category column is key. The input and expected output for these use cases are deterministic, and are therefore not usefull.

Since Torxakis does not have floats, our model will not test for this. This means that no Builtin Function is tested, since they only make sense when using floats. We will not use Builtin Variables in expressions, since all Builtin Variables are floats too. But we do test if the Builtin Variables are defined correctly, by calling them without other operators.

2.2 MBT Modeling

Ons model is genaamd

`_text.txs`

Expression Expressies kunnen bestaan uit elke combinatie van +,-,* en getallen. Er wordt gecheckt of de SUT de juiste uitkomst van deze expressie geeft.

1/0 Er wordt gecheckt of bij deze input de SUT daadwerkelijk een error geeft.

equality Ons model genereert twee expressies en vraagt aan de SUT of de eerste gelijk, groter of kleiner is dan de tweede. Het checkt of het antwoord klopt.

exit Dit test of het commando 'exit' aan de SUT geven, 'bye!' teruggeeft en dat de SUT vervolgens niet meer reageert op input.

pi Kijkt of de definitie van pi 3.141592653589793 is

e Kijkt of de definitie van e 2.718281828459045 is

function definition Genereert een van de drie voorgedefinieerde functies en onthoudt die.

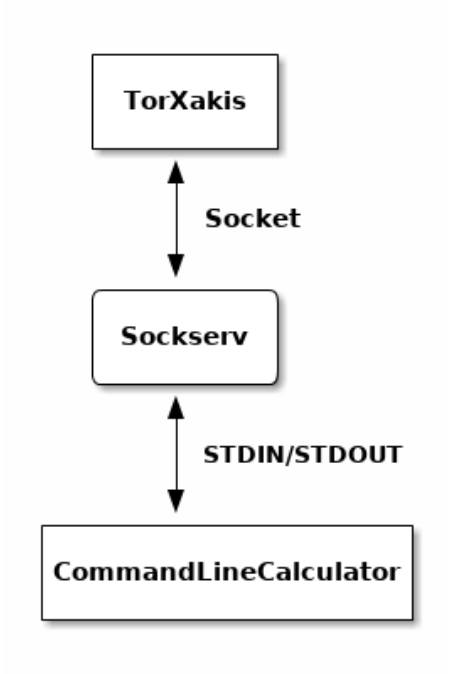
function application Er zijn hier twee opties. Of de functie is gedefinieerd, of niet. Als de functie gedefinieert is, wordt gecheckt of de SUT het juiste antwoord geeft. Als dat nog niet zo is, wordt gecheckt of de SUT de juiste error geeft.

variable definition Genereert een van de drie voorgedefinieerde variable definitions en onthoudt die.

variable application Checkt of de SUT het juiste antwoord geeft als de variable gedefinieerd is, en als het niet nog niet gedefinieerd is, checkt het of de juiste error gegeven wordt.

2.3 MBT Test Environment

Out test environment consists or our main Tool, Commandlinecalculator.jar, Torxakis and a proxy.



Our main application, Commandlinecalculator is a command line tool. It reads from STDIN en outputs to STDOUT and STDERR

SockServ.jar is a proxy between Torxakis and the CommandLineCalculator. It opens a socket and passes messages through to the Commandlinecalculator. In addition it simplifies the output, showing only the final result. This simplified allows us to use a less verbose model in torxakis

2.4 MBT Testing

All tests succeed. If the exit action is executed, following steps are 'quiessence', as should be.

Right now our TorXakis model generates nested arithmetic expressions and compares the evaluation done by Torxakis with the result given by our SUT.

Function and variable assignments are only tested shallowly yet since this would require a model potentially more than our SUT.

2.5 Deliverable

The code for both the SUT, the wrapper and the Torxakis models is publicly avaible at `https://github.com/Witik/CommandlineCalculator`.

To start the proxy service and SUT simply run the Sockserv.jar with as arguments the port to run on and command that you would otherwise execute. If you are in the root directory of the git project you can use the command

```
java -jar SockServ.jar 7890 java -jar target/CommandlineCalculator-1.0-jar-with-depe
```

Then run the torxakis tests with the following command from the torxakis directory

```
cmd.exe /c torxakis.bat /parth/to/git/root/torxakis models/_test.txs
```