Prof. Dr. A. Podelski,
Dr. B. Westphal

Sommersemester 2017

Softwaretechnik/Software Engineering

http://swt.informatik.uni-freiburg.de/teaching/SS2017/swtvl

Exercise Sheet 6

Early submission: Wednesday, 2017-07-26, 12:00 Regular submission: Thursday, 2017-07-27, 12:00

Exercise 1 – Black-box Testing (7/20 + Bonus)

For this task, we have developed an implementation of the shipping costs calculator of Exercise Sheet 3. The specification for the calculation is the decision table for shipping cost calculation without the COD rule (R1) given in the tutorial slides (Slide 22).

The implementation has the following specification (pre- and postcondition):

- The dimensions allowable for the package are input in whole centimeters and may range from 1cm to 250cm.
- The weight of the package is input in whole grams and may range from 1g to 32000g.
- The type of the destination address is given as a string, it can be 'metro', 'interm' or 'rural'.
- The program calculates the price in cents and prints it on the screen.

The users of the program observed some calculations being performed incorrectly. Unfortunately, they cannot remember which inputs caused the program to misbehave. You are now in the role of the test engineer and are requested to create a test suite. In particular, we are interested in determining which rules of the decision table have been implemented incorrectly.

(i) Give a **test suite** to check whether the program correctly implements all rules of the decision table. I.e., whether the program calculates the shipping price as specified when operated inside its specification. Describe the strategy you used to select the test cases. (5)

Submit a test script according to the instructions on Page 2 (not embedded in PDF).

Hint: You can assume that the project leader allotted a time budget for the creation of the test suite equivalent to 5 exercise sheet points. Submit a number of test cases that is reasonable to achieve and to explain within that time budget.

(ii) Your tutor will execute the tests in your test suite using your test script. There are the following possibilities:

- Your test suite finds one rule that is implemented incorrectly. (1 Bonus)
- Your test suite finds a second rule that is implemented incorrectly. (5 Bonus)
- Your test suite finds a third rule that is implemented incorrectly. (100 Bonus)
- Your test suite finds more rules that are implemented incorrectly. (1000 Bonus each)

(iii) What is the number of test cases required to *exhaustively* test the shipping calculator *inside its specification*?

Assume that you can execute around 1000 tests per second. How many seconds (hours? days? ...) would it then at least take to exhaustively test the shipping calculator using one (single core) CPU? (2)

Instructions for the Testing Exercise

Before you begin, you need to request a team ID to your tutor by mail. We have created a separate binary to test for each team. Your tutor will assign a number from 0 to 99. The team ID will be used to evaluate your results on the particular implementation of the shipping calculator.
In order to create a test script, login to one of the Linux machines of the computer pool and perform the following steps:

- Create a directory where you would like to store your results.
- Execute the installation script as follows:
  `/home/westphal/testing/install.sh ID`
  where `ID` is the group ID that you received from your tutor. The script will create the file `testsuite.sh` on the current directory.
- Use your favorite editor to open that file and insert one line per test case at the end of the file using the folloowing format:

  ```
  do_test W H L Wt Type Exp
  ```

  where:
  - `W` is the width of the package in cm.
  - `H` is the height of the package in cm.
  - `L` is the length of the package in cm.
  - `Wt` is the weight of the package in grams.
  - `Type` is the type of the shipping address: 'metro', 'interm' or 'rural'.
  - `Exp` is the expected result of the test.
- Save the file and submit it.

Running the tests (optional)

Note that task (i) only requires you to submit your test script. However, if you would like to execute the test suite yourself, the binaries for testing are accessible. You may execute them at your discretion.

- To run the test cases in your test suite, execute the command:
  `./testsuite.sh`
  on the host: `login.informatik.uni-freiburg.de`
- The script will execute your test suite and report the results.

Exercise 2 – Testing & Coverage Measures (6/20 + 3 Bonus)

Consider the Java function `convert` shown in Figure 1 on Page 3. It is supposed to convert the string representation (decimal, base 10) of an integer to the represented integer value. For example, the string "123" is converted to the integer `123` and the string "-378" is converted to the integer `-378`.
The following exceptional cases should be considered, i.e. corresponding exceptions should be thrown:

- The input string `str` has strictly more than 6 characters.
- The input string has at most 6 characters, and one of them is not a digit, i.e. from {'0', ..., '9'} (the first character may in addition be a minus ('-')).
- The input string has at most 6 characters, all of them digits (possibly a '-' in front) and the denoted integer value is outside the range $[-32768, 32767]$.

If none of the exceptional cases applies, let $c_0, \ldots, c_{n-1}$, $0 \leq n$ be the 1st, ..., n-th character in `str`. The expected return value is $\sum_{i=0}^{n-1} c_i \cdot 10^{n-i-1}$ if the first character c_0 is not '-' and $-\sum_{i=1}^{n-1} c_i \cdot 10^{n-i-1}$. if the first character c_0 is '-'. (Unlike other implementations of string-to-integer conversions, this one should return 0 for the empty string and the string "-".)

(i) Give an **unsuccesful test suite** for `convert` that achieves 100% *statement coverage* and 100% *branch coverage*.

Recall that submissions need a presentation that is comprehensible enough to be evaluated. Make sure that it is easy to see what the results are, which statements and branches are covered by each test case, etc. (5)

(ii) Modify your test suite from (i) such that it is still **unsuccesful** and still achieves 100% *statement coverage* but achieves **strictly less** than 100% *branch coverage*. (1)

(iii) Consider the expression $expr \equiv x \geq 0 \wedge (y = \texttt{'a'} \vee (z < 0 \wedge v)) \vee \neg(w \neq 100)$ for inputs x:`int`, y:`char`, z:`int`, v:`bool`, w:`int`. Give a test suite that achieves the maximum possible *term coverage* for $expr$. What is the maximum possible term coverage? Justify your answer. (3 Bonus)

```
1  int convert(char[] str) throws Exception {
2          if (str.length > 6)
3                  throw new Exception("Length exceeded");
4          if (str.length == 0)
5                  return 0;
6          int number = 0;
7          int digit;
8          int i = 0;
9          if (str[0] == '-')
10                 i = 1;
11         while (i < str.length){
12                 digit = str[i] - '0';
13                 if (digit <= 0 || digit > 9)
14                         throw new Exception("Invalid character");
15                 number = number * 10 + digit;
16                 i = i + 1;
17         }
18         if (str[0] == '-')
19                 number = -number;
20         if (number > 32767 || number < -32768)
21                 throw new Exception("Range exceeded");
22         return number;
23 }
```

Figure 1: Function `convert`

Exercise 3 – Grading Test Suites (2/20)

Assume there is a task to create an unsuccessful test suite for the function shown in Figure 2a on Page 4 (a complicated implementation of the function $\min(n, 255)$). The test suite should achieve 100% branch and statement coverage and document how this coverage is achieved. Consider the proposed solution shown in Figure 2b. Branches and statements are denoted by b_1, s_1, s_2, respectively, counted from top to bottom.

- Assume that a correct solution would be worth 4 points. There is a proposal to grade the proposed solution shown in Figure 2b with 0 points. Argue in favour of grading the proposed solution shown in Figure 2b with 0 points.

```
1 int min255( int n ) {
2   if (n < 255)
3     return n;
4   else
5     return 255;
6 }
```

(a) Function `min255()`.

input	$b_1/true$	s_1	$b_1/false$	s_2
0	✔	✔	✘	✘
255	✘	✘	✔	✔

(b) Proposed solution.

Figure 2: Test unit and test suite.

Exercise 4 – Verification with PD Calculus (5/20 + 5 Bonus)

Consider the program `multiply` shown in Figure 3. It is annotated with pre- and postconditions and implements integer multiplication as successive addition. The operands are x and y and the result is stored in the variable *result*.

$\{x \geq 0 \wedge y \geq 0\}$

```
result = 0;
i = 0;
while (i < y) do
  result = result + x;
  i = i + 1;
od
```

$\{result = x \cdot y\}$

Figure 3: Program `multiply`.

(i) **Give an invariant** for the while loop that enables you to prove the correctness of the program. Apply the rules of the *proof system PD* to **derive a proof** that your invariant is a loop invariant. (2)

(ii) Apply the rules of the *proof system PD* to **derive a proof** that `multiply` is *partially correct*.

Note: Please specify which rules or axioms you use at every proof step. Can you reuse the result of task (i)? (3)

(iii) Rewrite the program `multiply` as a C function with the following signature:

```
int multiply( int x, int y );
```

For a start, assume that `multiply` is used in a bigger program where it is only called with values for x between 0 and 15 and values for y between 0 and x, i.e. in the considered bigger program, all callers actually guarantee the precondition

$$\{0 \leq x \leq 15 \wedge 0 \leq y \leq x\}.$$

Annotate your program with the precondition, the postcondition, and the loop invariant (and whatever else you consider necessary) using the VCC syntax for annotations and use VCC[1] to **verify** your annotated program. (2 Bonus)

Hint: Declaring `result` and `i` as local variables of the function makes the task a bit easier. Otherwise, with, e.g., `result` being a global variable, you would need to add a clause `_(writes &result)` to your function declaration.

[1] `http://rise4fun.com/vcc`

To enable the tutors to reproduce your results, your annotated C code (with appropriate comments or explanations, if necessary) needs to be a separate part of your submission (not inside PDF).

(iv) Discuss the relation of the verification results you obtained from VCC and your result from task (ii). Can you expect VCC to confirm your results? (1 Bonus)

(v) If we verify the considered C program for pre-conditions

$$\{0 \leq x \leq N \wedge 0 \leq y \leq x\}$$

with $N = 15, 150, 1500, 15000$, we observe that the verification time reported by VCC is approximately the same.

Given that a VCC verification for N corresponds to more than $\frac{1}{2}N^2$ test cases (which is a lot for $N = 15000$), is that measurement plausible? Explain. (1 Bonus)

(vi) Assume you are unsure about your proof from task (ii) and you would like to confirm the result using VCC.

Modify the C program such that it in particular considers the original precondition (cf. Figure 3) and try to verify it using VCC. Describe what you expect to be the outcome of this experiment and what the results of the verification with VCC were. Give an interpretation, in your own words, of the output of VCC. (1 Bonus)

Exercise 5 – Verification with VCC (5 Bonus)

Recall the shipping calculator program from Exercise 2. After finding errors through testing, the developers have fixed the program. Now the test suite is unsuccesful. Having seen that there is a very large number of possible inputs for the program, now we would like to use verification –instead of testing– to make sure that the program works correctly.

In the file `calculate_price.c`, you will find the implementation of the function `calculate_price`. It takes as input the following parameters:

- `actual_weight`: an integer between 1 and 32 representing the actual weight of the package in kilograms, after conversion from grams.
- `v_s`: an integer between 1 and $250 \times 250 \times 250/2500$ representing the volumetric weight of the package when using the factor for small packages.
- `v_p`: an integer between 1 and $250 \times 250 \times 250/5000$ representing the volumetric weight of the package when using the factor for parcels.
- `address_type`: a value of type `addrtype` representing the type of the destination address of the package.

(i) Annotate the function `calculate_price` with **preconditions** for the allowable ranges of the parameters, and with **postconditions** that ensure that the price calculation for rules R9 and R10 of the decision table from Tutorial 3 (parcels for rural addresses) are correctly implemented. Verify the program with VCC, document and discuss your results. (3 Bonus)

Hint: you will find that we have already declared the function `max`. You may also use that function in your specification. Make sure to submit your source code together with your solutions.

(ii) Add aditional **postconditions** to `calculate_price` to verify that all other rules of the decision table (R2-R8) are also correctly implemented. Verify the program with VCC, document and discuss your results. (2 Bonus)

Hint: this exercise is somewhat challenging since the lecture cannot introduce all peculiarities of the tool VCC. If you want to give it a try, don't hesitate to ask your tutor for help if you get stuck.