# Software Testing LTAT.05.006

# Lab 5: Automated Unit Test Generation

Inst. of Comp. Science, University of Tartu

Spring 2020

Submission deadline: Lab reports must be submitted within seven days. For example, if your lab takes place on a Tuesday, then you have to submit your report no later than on the following Monday, 23:59 hours.

- Late submission policy:
  - 50% of the total marks deducted for submission up to 24 hours late
  - 100% of the total marks deducted for submission more than 24 hours late
- Group: There should be a maximum of two members in a group. Answers should be your own group work, explained in your own words. If you work in a group of two students, make sure to mention the names of both students in the submitted lab report.
- Maximum amount of points is ten (10).

## 1 Introduction

### 1.1 Unit Testing

Unit testing is a part of the testing process, which focuses on testing the implementation of the software. Unit testing is the lowest level of testing and is often the responsibility of the programmer. There are many tools available to manually write unit tests, such as JUnit but there also exist tools that automatically generate unit tests, such as Randoop.

### 1.2 Learning Objectives

The goal of this lab is to introduce automated test generation. Writing tests manually can be difficult and time-consuming. Automating test generation will speed up the software development process. It is important to understand the strengths and weaknesses of the tools you are using. Automatically generated tests can sometimes include test cases the developer did not think of but also test cases the developer would not think of.

## 2 Systems Under Test

### 2.1 NextDate class

The first system under test is the NextDate class from Lab 4. It is a simple class that has the purpose to return a date that follows the input date.

### 2.2 POS System

The second system under test is the point-of-sale system from the course Software Engineering (LTAT.05.003). It is a small application to manage items in stock and make sale transactions. The stock is managed by the class InMemorySalesSystemDAO and purchases are made using the ShoppingCart class.

## 3 Randoop

### 3.1 Introduction

Randoop is a tool used to automatically generate unit tests. The tests generated by Randoop are classified either as *error-revealing* tests or as *regression* tests.

Error-revealing tests are tests which have found a bug. Sometimes the errors revealed are not too serious and it is up to the programmer to decide to fix the bug. Randoop can expect exceptions if they are caught properly in the code. Randoop will also expect a NullPointerException, if null is directly passed as an argument to a method.

Regression tests are tests which passed and therefore did not find any errors. The purpose of regression tests is to alert you to changes in the program’s behavior after changing the code. It is important to analyze those tests in the regression test suite that are failing as a result of changing the code. If your new code is incorrect and has introduced a fault, the code should be fixed. However, if the changes were intentional, then the test suite is out of date and should be discarded. This may be because the tests were too sensitive to changes. After discarding the old (regression) test suite, a new test suite should be generated.

Figure 1 describes the general test generation and execution workflow when using Randoop.

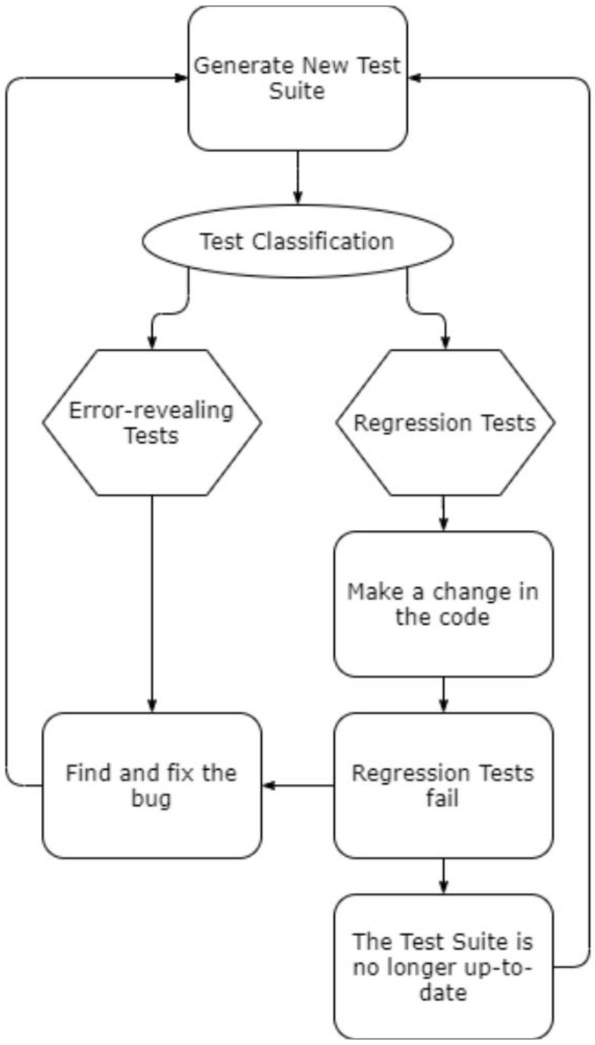

Figure 1. Test Generation and Execution Workflow

## 3.2 Setup

In this lab, we are generating unit tests using Randoop. To download the latest version of Randoop, go to: https://github.com/randoop/randoop/releases/latest. Unzip the downloaded file, inside you should see three .jar files and the readme file.

Randoop requires **Java 8 (x64)** to work, which can be downloaded here: https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html.

Choose the JDK which corresponds to your operating system and install it.

## 3.3 Usage

The basic command to run Randoop is:

*java -cp “C:\Path\To\Randoop-all.jar” randoop.main.Main gentests --testclass=’C:\Path\To\Class’*

If your JAVA_HOME Environment Variable is not set to Java 8, you will have to manually specify the correct path:

*“C:\Path\To\Java\8\bin\java” -cp “C:\Path\To\Randoop-all.jar” randoop.main.Main gentests --testclass=’C:\Path\To\Classes\Class’*

It may be necessary to specify your project’s output folder in the classpath variable. Then it will not be necessary to specify the path to the class under test in the --testclass flag.

*“C:\Path\To\Java\8\bin\java” -cp “C:\Path\To\Randoop-all.jar;C:\Path\To\Classes” randoop.main.Main gentests --testclass=’package.name.Class’*

In the process of generating tests, Randoop will generate test sequence .class files. These are useful for a larger production environment to guarantee that identical test sequences would be generated when generating a new test suite. For the purposes of this lab, they may be deleted.

### 3.3.1 Basic Troubleshooting

You may run into some technical issues, below are some examples to help you.

Error: *Unable to load class "MyExampleClass" due to exception: java.lang.NoClassDefFoundError: MyExampleClass.* To fix this error change MyExampleClass to ee.ut.cs.swt.example.MyExampleClass

Error: *java.lang.IllegalStateException: Cannot find the Java compiler. Check that classpath includes tools.jar.* This error is thrown when you are using Java 9 or above, use Java 8 instead.

## 3.4 Parameters and Flags

To specify parameters, flags can be added to the command. The example in section 3.3 uses the *--testclass* flag. The main flags required for this lab are given below.

*--testclass=’C:\Path\To\Class’*, for example ‘java.util.Set’

*--classlist=’C:\Path\To\Classes.txt’* with this flag you can specify a text file which lists all the classes to be tested.

*--methodlist=’C:\Path\To\methods.txt’* with this flag you can specify the methods to be tested in a file. Each method has to include a fully qualified signature. For example

*java.util.Collections.rotate(java.util.List,int)*. A full example file can be found here: https://randoop.github.io/randoop/manual/method_list_example.txt

*--checked-exception=enum*, the value of this parameter is selected from {ERROR, EXPECTED, INVALID}. It is used to specify how to classify exceptions which are already handled in the code. ERROR will classify the tests as error-revealing, EXPECTED will classify the tests as regression tests, INVALID will discard the tests. The default value is EXPECTED.

*--npe-on-null-input=enum,* the value of this parameter is selected from {ERROR, EXPECTED, INVALID}. It is used to specify how to classify NullPointerExceptions if null is passed as an argument. The default value for this parameter is EXPECTED.

*--time-limit=int* It is used to give an amount of time in seconds, to generate tests. The default value of this parameter is 100.

*--output-limit=int* The amount of regression- and error-revealing tests to generate. The default value of this parameter is 100000000.

*--maxsize=int* This is used to specify a maximum amount of statements in a test. The default value of this parameter is 100.

*--junit-output-dir='C:\Path\To\Output* This flag is used to specify the test file output directory

*--literals-file=*'C:\Path\To\File'. This flag is used to give a file containing literal values to be used as inputs to methods under test. Randoop uses the following values by default:

byte: -1, 0, 1, 10, 100

short: -1, 0, 1, 10, 100

int: -1, 0, 1, 10, 100

long: -1, 0, 1, 10, 100

float: -1, 0, 1, 10, 100

double: -1, 0, 1, 10, 100

char: '#', ' ', '4', 'a'

java.lang.String: "", "hi!"

The format of the literals file is specified at the appendix. You can read more about it at https://randoop.github.io/randoop/api/randoop/reflection/LiteralFileReader.html.

The full Randoop manual can be found here: https://randoop.github.io/randoop/manual/

## 4 In-class Work

4.1 Download the NextDate code from the course page and unzip it. Open the project in your IDE. You will have to compile the project, because you need to provide Randoop the class files.
4.1.1 Move the existing NextDateTest.java file to a separate folder, so that there are only error-revealing (if found) and regression tests in the 'test' folder. NB! For your homework, you will need to compare code coverage between automatically generated and manually written tests, so do not delete the NextDateTest.java file.
4.2 Use Randoop to generate tests for the NextDate class. Take a screenshot of the Randoop command. You will need to submit it as part of the homework for this lab. Move the generated tests to the 'test' folder. NB! It is expected that you experiment and get to understand Randoop at this subtask.
4.3 It may be necessary to specify the correct package for the tests, IntelliJ might not recognize the necessity for the correct package, since the file containing the tests is fairly large. This can likely be fixed by adding *package ee.ut.cs.swt.nextdate;* to the start of the files containing the tests.
4.4 Execute the Maven goal 'test'. If you run into issues here, make sure that the Java 8 JDK is used throughout.
4.5 Show the JaCoCo code coverage report to the lab supervisor.

## 5 Homework Tasks and Reporting

### 5.1 Task 1: Randoop Code Coverage of NextDate

5.1.1 If you were not present in the lab session, you will first have to complete section 4 above (i.e., using the default parameter settings of Randoop). In addition to the test suite generated with default Randoop parameters, generate test suites using the following values for the time-limit parameter: 3, 30, 60. Other parameters keep default values.

Reporting Item T1.1:

Take a screenshot of each Randoop command and include them in your report. Make sure that your screenshots are readable. The screenshots should contain the commands you enter and the output from Randoop.

In addition, report the results from this task in a table using the format as shown in the appendix in the example table. You do not need to submit the generated tests, but you must store them somewhere so you can show them on request.

5.1.2 Explain why you got identical coverage results in task 5.1.1.

Reporting Item T1.2:

Report your explanation in free text format.

5.1.3 Improve code coverage by using additional Randoop parameters (hint: here it's best to use a literals file). Make at least 2 attempts to improve code coverage.

Reporting Item T1.3:

Copy the table from Reporting Item T1.1 and extend the table with two new rows corresponding to the two attempts you made to improve code coverage.
In addition, include a screenshot of a Randoop-generated test case with a valid input date in your report.
In addition, create a folder (named T1.3) with the following information:

- Include the test suite (attempt 1) generated in this task.
- Include the test suite (attempt 2) generated in this task.
- include the two literals files used in the two attempts that you report.

5.1.4 Randoop did not generate error-revealing tests. This could be the case because the program is correct. However, assuming the program was not correct, what could in that case be the reason for not generating an error-revealing test?

Reporting Item T1.4:

Report your explanation in free text format.

5.1.5 When writing manual tests, one usually achieves higher coverage than with automatically generated tests. Probably, when you wrote tests for the NextDate program in Lab 4, you had a higher coverage than with using Randoop. For the purpose of comparison, report your own highest coverage results from Lab 4, where you wrote the tests manually. If you did not do lab 4, you may ask your fellow students for their code coverage report file (a part of the homework of lab 4) on the course Slack channel. In addition to reporting the code coverage, answer the following question: What is the benefit of using Randoop although you (most probably) achieve lower code coverage with Randoop?

Reporting Item T1.5:

Copy the table from Reporting Item 3 and extend the table with one new row showing the code coverage that you achieved in Lab 4.
In addition, report the answer to the question about the advantage of using Randoop despite having lower code coverage.

## 5.2 Task 2: POS Testing

5.2.1 Download the project file from the course page and unzip it. Open the project in your IDE. Generate a test suite using Randoop for the POS system using a reasonable amount of time for the time-limit parameter. Make sure that the tests are in the test folder of the project. Then run the tests using your IDE. It may be necessary to specify the correct package for the tests by adding *package ee.ut.math.tvt.salessystem;* to the start of all the files containing the tests. Keep the test suite, since you have to run it again in task 5.2.3.

Reporting Item T2.1:

Take screenshots of all runs of Randoop and include them in your report. Again, make sure the screenshots are of good quality.
In addition, create a folder (named T2.1) with the following information:

- Include the test suite generated in this task.
- If you used a text file that lists the classes to be tested, include the file

5.2.2 If you set the time-limit parameter appropriately, Randoop generates error-revealing tests. Analyze the tests and explain the failure(s) discovered. Point out the block or line of code which caused the exception. You will not have to fix the bug(s) found, only report them.

Include a Randoop-generated error-revealing test case in the report

Reporting Item T2.2:

Provide a screenshot of the error revealing test case.
In addition, copy the block or line of code in the POS application that caused the exception into your report and explain why the exception was thrown (i.e., explain what is wrong in the code).

5.2.3 Implement the TODO in the ShoppingCart class by uncommenting the code under the text 'TODO'. The purpose of the TODO part is to increase the quantity of the item in the shopping cart if the stock item is already in the cart.
Run the test suite generated in task 5.2.1 again on the updated code (i.e., after uncommenting the TODO code). Look at some of the failing tests in the regression test suite and explain whether the tests reveal a bug you've introduced or whether the test suite has become outdated. You may need to investigate the code more in-depth.

Reporting Item T2.3:

In the report, provide a screenshot of the error revealing test case.
In addition, provide an explanation whether the tests reveal a bug you've introduced or whether the test suite has become outdated.

5.2.4 Run Randoop on the updated code and generate a new test suite. Find a test case which contains the same key steps as an error revealing test from task 5.2.3 (it doesn't have to be the same that you used in reporting item T2.3) and compare the two tests and explain what is different. In particular, explain why Randoop handles NullPointerExceptions differently in some cases. (Hint: Look at how Randoop handles the exception.)

Reporting Item T2.4:

In the report, provide a screenshot of an error revealing test case from task 5.2.3 and a screenshot of a corresponding test in the newly generated test suite.
In addition, explain what is different between the two tests for which you provide the screenshots.
In addition, create a folder (named T2.4) with the following information:

- Include the test suite generated in this task.

5.2.5 Assume that you want Randoop to classify the passing tests (and other essentially similar tests) from task 5.2.4 as error-revealing. Generate a new test suite, which classifies these tests as error-revealing. Include a Randoop-generated error-revealing test case in the report. Hint: Understand the parameters that define how Randoop handles exceptions.

Reporting Item T2.5:

In the report, provide a screenshot of an error revealing test case contained in the newly generated test suite.
In addition, provide a screenshot that shows the new parameter settings you used to generate the new test suite.
In addition, create a folder (named T2.5) with the following information:

- Include the test suite generated in this task.

## 6 Submission and Grading

### 6.1 Submission

For the homework, you have to submit a ZIP-file at the course lab submission page, that includes your report and all the folders specified in the reporting items in Section 5.

### 6.2 Grading

For this lab, you can receive 10 points in total:
1 point for lab attendance
5p for Task 1

- 1 point for T1.1
- 0.5 points for T1.2
- 2 points for T1.3
- 1 points for T1.4
- 0.5 points for T1.5

4p for Task 2

- 0.5 points for T2.1
- 1 point for T2.2
- 1 point for T2.3
- 1 point for T2.4
- 0.5 points for T2.5

## Appendix

### Randoop Literals File Template

START CLASSLITERALS
CLASSNAME
classname
LITERALS
type:value
...
type:value
END CLASSLITERALS

### Table Template for Homework Task 1

| Number of tests generated | Instruction & Branch Coverage | Maven test goal execution time | Additional parameters used* | Randoop log screenshot file name |
|---|---|---|---|---|
| 100 | IC 34%, BC 23% | 3.536s | --output-limit=100 | log1.jpg |
| | | | | |

* If none of the default parameter settings has been changed, then write ‘none’.