Exercise 13: Random Testing

The objective of this exercise is to explore strategies and tools for random testing in order to understand the benefits and drawbacks of this approach.

13.1 Generating Test Cases with Randoop (2 points)

Randoop is an example for an automatic unit test generator for Java. It implements a random test generation strategy to automatically create unit tests for Java classes (see *Randoop: Feedback-directed Random Testing for Java*[1] by C. Pacheco and M.D. Ernst, OOPSLA 2007). In that way, Randoop is able to create an extensive set of regression tests, i.e., test cases that check for defects in an existing implementation due to side effects of changes or enhancements. Regression-testing with Randoop is characterized by (a) generating the test cases from a correct version of the tested software system and (b) running these test cases against a different and likely defective version (e.g., a new release).

Instructions

Use Randoop to generate test cases for the class *RingBuffer,* the example introduced in the previous assignments. Run the generated test cases against the modified (mutated) version of *RingBuffer* and assess how many of the mutations can be detected. Consider following hints:

- ☐ Go to the update site http://randoop.googlecode.com/hg/plugin.updateSite/ to install the Randoop Eclipse Plugin; installation instructions and additional information can be found at http://randoop.googlecode.com/hg/plugin/doc/index.html.
- ☐ Create a new Eclipse Java project named SQE13-*Lastname*_RingBufferRandoop; replace *Lastname* by your last name. Add a source package *randoop*, and copy the implementation *RingBuffer.java* into this package. Add JUnit 4 to the build path.
- ☐ Right click on *RingBuffer* and select *Run As > Randoop Test Input*; set the output folder to *src*, use *randoop* as package for the generated test cases, and name the test *RingBufferTest*. Set the time limit to, e.g., 30 sec.
- ☐ Once Randoop has successfully generated test cases (green bar in Randoop), run them with JUnit and make sure they pass (green bar in JUnit).
- ☐ Use PIT to mutate the implementation of *RingBuffer* and to re-execute the generated test cases. Document your results by taking a copy the PIT report showing the mutation score.

Submission

Submit the Eclipse project containing the *source code of the tests* (test cases and test suite) generated with Randoop, the *PIT report* and any other source code files you may have created.

[1] http://people.csail.mit.edu/cpacheco/publications/randoopjava.pdf

13.2 Using Randoop for Finding Bugs (3 points)

In the regression-testing scenario, Randoop is used to generate the test cases from an initial version of the software system that is assumed to be correct. The generated test cases are then used to detect defects in a new, modified version of this software system. But what if there defects exist in the initial version? Randoop can also detect such defects! Additional information is necessary to tell Randoop whether the behavior of the system is correct or faulty. Assertions part of the Java programming language can provide such information, see http://download.oracle.com/javase/1.4.2/docs/guide/lang/assert.html.

Instructions

- ☐ Copy the class *RingBuffer* to *RingBufferMutant* and manually introduce a defect. For example, comment out the statement *N--* part of the method *dequeue()*.
- ☐ Add Java assertions to the modified class *RingBuffer:* check correct internal states at the end of various methods of *RingBuffer*, e.g., in *isEmpty()* or *size()*. Use Randoop again to generate test cases. Since Randoop evaluates assertions, the introduced defect should now be detected during test case generation and a corresponding failing unit test will be created. Note:
  - *Randoop* will enable assertions per default. When running the generated tests you have to enable assertions via the command line option *-ea* (enable assertions) in the run config: *Run As > Run Configuration > Arguments > VM arguments*: add *-ea*.
  - *Read* the article *Using Assertions in Java* by *M. O'Gara et al*. Pay special attention to the section *Tips on Using Assertions*.

Submission

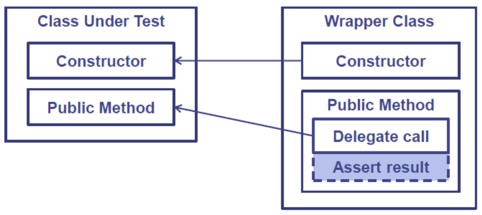
Submit the Eclipse project containing the modified class *RingBufferMutant* with the assertions and the *source code of the generated failing tests* exposing the defective modification. Note: Only the failing tests are required!

13.3 Including Assertions via Wrapper Classes (4 points)

Assertions are an important measure for finding bugs and assuring quality. They should already be included when writing code. For legacy systems, however, it is not always possible to modify/extend the existing code.

In this scenario a *Wrapper Class* can be use, which (1) delegates the method calls to the *Class Under Test* and (2) verifies the result using assertions.

Randoop is used on the *Wrapper Class*, so the assertions are evaluated when generating tests.



Instructions

- ☐ Create a wrapper class *RingBufferWrapper<Item>* for the class *RingBuffer<Item>*. The wrapper class should have the same set of public methods. Method calls should be delegated to the class RingBuffer.
  - Create a new class *RingBufferWrapper<Item>* and add a member variable of type *RingBuffer<Item>*. Right click on the type declaration and run *Source > Generate Delegate Metods...* Select a meaningful subset of the proposed delegate methods for

testing; do not include *equals(), hashCode(), ...*

  - Add a constructor to the class *RingBufferWrapper*. Use it to instantiate a new RingBuffer object.
  - For each delegator method of *RingBufferWrapper*, add *assert* statements that implement checks for the correct handling of preconditions, postconditions and invariants. (Derive these checks from your understanding of how the *RingBuffer* should work; the actual implementation may deviate from your view due to defects in the implementation.)
  - Use Randoop (see above) to generate test cases. (If any asserts are violated, Randoop will indicate the problem and generate failing test cases in a separate folder.)
  - Run the generated tests and analyze the coverage of the class *RingBuffer*.

☐ Modify the class *RingBuffer* manually to introduce a defect. For example, comment out the statement *N*-- part of the method *dequeue()*. Will the modification be detected by Randoop?

Submission

Submit the Eclipse project containing the wrapper class *RingBufferWrapper* and the generated tests exposing the defective modification.

14.1 Adventures with Monkey Tests (1 points)

In the chapter *Adventures with Monkey Tests,* John Fodeh describes an automated random test approach, which is also known as "monkey testing". The approach aims at testing the reliability of embedded and Windows-based medical applications.

Reference: John Fodeh: Adventures with Monkey Tests*,* In: D. Graham and M. Fewster (Eds.) *Experiences of Test Automation: Case Studies of Software Test Automation*, Pearson Education, 2012

Instructions

Read the book chapter carefully and answer following questions. Document each of your answers in a short paragraph in the text file *exercise14.txt.*

1. How does (dynamic) monkey testing overcome the typical limitations of (static) automated regression testing? Answer this question by briefly addressing each of the limitations listed in the sections 24.2.1 to 24.2.5.

2. What does the metric *mean number of random operations between failures* describe and how is this metric interpreted to support test and release decisions?

3. What is – from your point of view – the biggest problem when using monkey testing in a real-world project? Explain your answer.

Submission

- Submit the text file exercise14.txt containing your answers.

General Submission Instructions

- Add a comment at the beginning of every source code file containing your full name and your student ID number.
- The Eclipse projects have to be completely self-contained. Do not use absolute paths to reference libraries or otherwise the submitted exercise cannot be compiled without build errors and will not be accepted.
- Submit the complete Eclipse project including all relevant additional files as specified via packing in a Zip archive. Adhere to the naming convention for projects: SQE##-*Lastname_Title* (## = exercise number, *Lastname* = your name, *Title* = exercise title).
- Upload the archive at the specified submission link on the elearning platform http://elearning.fh-hagenberg.at/ until the specified date and time.