THE UNIVERSITY OF NEWCASTLE AUSTRALIA

SCHOOL OF INFORMATION AND PHYSICAL SCIENCES

School of Information and Physical Sciences

SENG3320/6320 Software Verification and Validation

Assignment 2

This assignment is to be completed in groups.

*Due on **Friday, 6 June 2025, 11:59pm**, electronically via the "Assignment 2" submission link in Canvas.*

Total 100 marks
(Weight 25% of the overall course assessment)

Aims

The aim of this assignment is to develop students' practical understanding of automated test data generation techniques, with a particular focus on fuzz testing, symbolic execution, and mutation testing. Students will learn to design and implement fuzzing tools to discover software vulnerabilities, analyze coverage effectiveness, and evaluate test suite quality.

Project Description

Part 1 (40 marks):

Fuzz testing (also known as random testing) is commonly used to discover software crashes by generating a large amount of random input data. It is a cost-effective alternative to more systematic testing techniques. In this assignment, you are required to apply fuzz testing to evaluate a KWIC program. The KWIC (Key Word in Context) problem is defined as follows:

The KWIC [Key Word in Context] index system accepts an ordered set of lines; each line is an ordered set of words, and each word is an ordered set of characters. Any line may be "circularly shifted" by repeatedly removing the first word and appending it at the end of the line. The KWIC index system outputs a list of all circular shifts of all lines in alphabetical order."

The KWIC system is a real system and is widely used in keyword in context indexes for libraries.

Example

Consider the following three book titles (lines):

– Pattern-Oriented Software Architecture

– Software Architecture

– Introducing Design Patterns

The KWIC system produces the following output:

– Architecture Software

– Architecture Pattern-Oriented Software

– Design Patterns Introducing

– Introducing Design Patterns

– Patterns Introducing Design

– Pattern-Oriented Software Architecture

– Software Architecture Pattern-Oriented

– Software Architecture

Users can now quickly search for book titles that contain phrases such as “Software Architecture” or “Design Pattern”.

For this assignment, you are provided with a Java implementation of the KWIC problem (*KWIC.class*), which can be found on Canvas. The program is executed using the following command:

java KWIC input.txt

where input.txt is a plain text file containing the input book titles (each line is a title).

You are required to apply fuzz testing to generate test inputs aimed at crashing the KWIC program. Specifically, you must:

a) Design and develop a fuzz testing tool that can feed a large volume of random data (e.g., random book titles such as “a8h h19%p”) into the KWIC program in an attempt to trigger crashes. **(20 marks)**

Note: Marks for this component will depend on the sophistication of your tool. A feedback-guided fuzzing tool (e.g., coverage-guided) can receive up to 20 marks if successfully implemented. A non-feedback-based (i.e., pure random) fuzzer can receive a maximum of 10 marks if correctly implemented and functional.

b) Perform the fuzz testing, record the number of unique crashes (i.e., crashes with different exception messages or occurring at different program locations/line numbers), and capture the input that causes each crash. **(10 marks)**

c) Write a test report describing the design of your fuzzing tool, the test environment, examples of test cases, and a summary of the test results (particularly the number of unique crashes detected). **(10 marks)**

Part 2 (60 marks):

In this assignment, you will experiment with **symbolic execution**, **mutation testing**, and **fuzz testing** techniques. Note that this part requires code implementations in both C and Java. Specifically, question (a) requires implementation in C, while questions (c) and (d) require implementation in Java.

The tasks for this part are broken down as follows:

a) Symbolic execution **(15 marks)**:
Apply KLEE to generate test data for the Triangle program shown below (KLEE can be downloaded from: https://klee-se.org/releases/).

```
/* The Triangle program, which determines if three inputs specify an equilateral
triangle, an isosceles triangle, an ordinary triangle, or a non-triangle. */
void triangle(int a, int b, int c) {
    if ((a + b > c) && (a + c > b) && (b + c > a)) {
        if (a == b || a == c || b == c) {
            if (a == b && a == c) {
                printf("equilateral triangle.\n");
            } else {
                printf("isosceles triangle.\n");
            }
        } else {
            printf("triangle.\n");
        }
    } else {
        printf("non-triangle.\n");
    }
    return;
}
```

b) Control-flow coverage **(5 marks)**:
Compute the control-flow coverage of the test data generated by KLEE, including statement coverage, branch decision coverage, condition coverage, condition/decision coverage, and multiple condition coverage.

c) Fuzz testing **(15 marks)**:
Apply coverage-guided fuzz testing to generate test data for the Triangle program. Compare the test results of fuzz testing and symbolic execution in terms of control-flow coverage achieved and the time taken to execute the test suite. It is sufficient to consider the coverage metrics listed in question (b).

d) Mutation testing **(15 marks)**:
Apply at least two mutation operators to the Triangle program and perform mutation testing:
i) Mutate the Triangle code using the selected mutation operators. At least 10 mutants should be generated. The mutations can be applied either manually or automatically using a simple mutation tool (no mark difference between approaches).
ii) Execute the test cases generated by symbolic execution against the mutants.
iii) Measure how many mutants are killed. If any remain, add new test cases to kill them.

e) Test report **(10 marks)**:
Write a test report describing your experimental design, steps taken, and results. The report should also include a comparison of the symbolic execution, mutation testing, and fuzz testing techniques used in this experiment.

Submission

All assignments must be submitted via Canvas. If multiple submissions are made, only the latest one will be graded. This assignment consists of two parts: (1) the project report and source code, and (2) the peer evaluation form (not graded). The project report and source code should be submitted by only one group member on behalf of the entire group (in a single .zip file). However, each group member must individually submit a peer evaluation form that reflects the contributions of all team members. The peer evaluation form must be submitted through the separate “Assignment 2 Peer Evaluation” submission link on Canvas.

Your submission (.zip file) must include the following:

(1) An assignment cover sheet.
(2) A PDF report that contains answers to all questions (excluding the source code of the tool implementations.
(3) A folder containing the code project, including: a). The complete project folder structure; b). A README file specifying: Instructions for compiling and executing the

tools; Any external libraries (and versions) or dependencies used; and The Java version and IDE (if any) used for development.

The mark for an assessment item submitted after the designated time on the due date, without an approved extension of time, will be reduced by 10% of the possible maximum mark for that assessment item for each day or part day that the assessment item is late. Note: this applies equally to week and weekend days.

Use of Generative AI tools

The use of generative AI tools is permitted for this assessment under the following conditions:

1) Generative AI tools may be used to assist with understanding concepts; however, they must not be used to directly generate answers to assignment questions. Directly copying and pasting AI-generated content into your report is considered plagiarism.

2) If generative AI tools are used (excluding cases where they are used solely for language assistance), you must document the following in a separate section at the end of your report:

- The name of the tool(s) used
- The prompts or queries submitted
- The corresponding AI-generated output (as copy & paste or screenshots)

3) Be mindful of the limitations of generative AI tools, such as hallucination (i.e., generating incorrect or misleading information). Always critically evaluate the output.

Plagiarism

All work must be your own. Plagiarism, including copying from other groups or unauthorized sources, will be handled in accordance with the university's academic integrity policy.