Using Clone Detection to Identify Bugs in Concurrent Software

Kevin Jalbert, Jeremy S. Bradbury ● Software Quality Research Group
Faculty of Science ● University of Ontario Institute of Technology ● Oshawa, Ontario, Canada
kevin.jalbert@uoit.ca, jeremy.bradbury@uoit.ca

1. Motivation

- It is challenging to develop high quality concurrent software.
  - **Why?** Multiple threads of execution can be scheduled in many different ways.
  - Tools already exist to test and debug these different schedules.
- We propose an active testing approach that is based on localizing the testing effort by first identifying potential bugs using clone detection.
  - **What is active testing?** Operates by first localizing the potential faults and then specifically testing them [1].

Research Goals:
1. Detect potential concurrency bugs in a software system using bug patterns based on previously detected bug.
2. Use the location of potential bugs to target future testing effort.

2. Background

- We use the clone detection framework **ConQAT** [2] as the foundation for detecting bug patterns in concurrent software because it is capable of finding exact, near-exact and gapped clones.
  - Exact clones are exactly the same textually.
  - Near-exact clones are the same in structure though textually they are different.
  - Gapped clones is a clone that has some statements added or removed.
- Locating potential bugs using clone detection makes it possible to localize the testing effort to a specific section of the source code.
  - **ConTest** [3] is used to insert random delays in the source code, increasing the coverage of the thread interleaving space during testing.

3. Defining Bug Patterns

What is a bug pattern?

- A bug pattern can consist of several fragments of code as well as rules about how the fragments interact to cause a bug.
- A potential bug is identified if all fragments are present and the rules are satisfied.

What is a bug pattern rule?

- A logical statement that describes when the interaction between fragments' terms may cause abnormal behaviour.
- Rules have limited expressiveness and only allow for simple relations using Boolean logical operators (||, &&, ==, !=, !).
- To increase the expressiveness of rules applying special properties to terms was necessary (e.g., IS_SYNCHED).

```
<bugPattern id="0"
<sourcePath="/bp/bug_pattern_0.xml">
  <type>Data race - no locks</type>
  <tester>John Smith</tester>
  <description>This bug exhibits inconsistent data of
      the obj object.</description>
  <solution>Synchronize both code
      fragments.</solution>
  <originalFragment sourcePath="/src/bp_code/
      bug_pattern_code_0_0.java" countLines="0"
      patternId="0" fragmentId="0">
    <term id="F0.obj" line="0" tokenPosition="3"/>
  </originalFragment>
  <originalFragment sourcePath="/src/bp_code/
      bug_pattern_code_0_1.java" countLines="0"
      patternId="0" fragmentId="1">
    <term id="F1.obj" line="0" tokenPosition="0"/>
  </originalFragment>
  <rule>(F0.obj == F1.obj && !F0.obj.IS_SYNCHED
      && !F1.obj.IS_SYNCHED)</rule>
</bugPattern>
```

Figure 1: Data race bug pattern in XML format

How are bug patterns created and managed?

- Bug patterns are identified by developers when a bug is discovered.
  - Bug patterns can describe either general or application-specific bugs.
- A bug pattern can be defined using our Bug Pattern Creator tool.
  - This tool creates an XML file that represents a bug pattern (see Figure 1).
  - A collection of user-created bug patterns can be managed using our Bug Pattern Creator tool.

[1] P. Joshi, M. Naik, C.-S. Park, and K. Sen, "CalFuzzer: An extensible active testing framework for concurrent programs," in *Proc. of the 21st International Conference on Computer Aided Verification (CAV 2009)*, 2009, pp. 675–681.
[2] E. Juergens, F. Deissenboeck, and B. Hummel, "CloneDetective – a workbench for clone detection research," in *Proc. of the 31st International Conference on Software Engineering (ICSE'09)*, 2009, pp. 603–606.
[3] O. Edelstein, E. Farchi, Y. Nir, G. Ratsaby, and S. Ur, "Multithreaded Java program test generation." *IBM Systems Journal*, vol. 41, no. 1, pp. 111–125, 2002.

4. Process

- Our approach finds sets of matching code fragments of bug patterns. Interaction rules are then applied to identify high- and low-potential bugs based on the interactions of the code fragments.
- Localization of testing effort around potential concurrency bugs is vital to increasing the efficiency and effectiveness of concurrency testing tools.

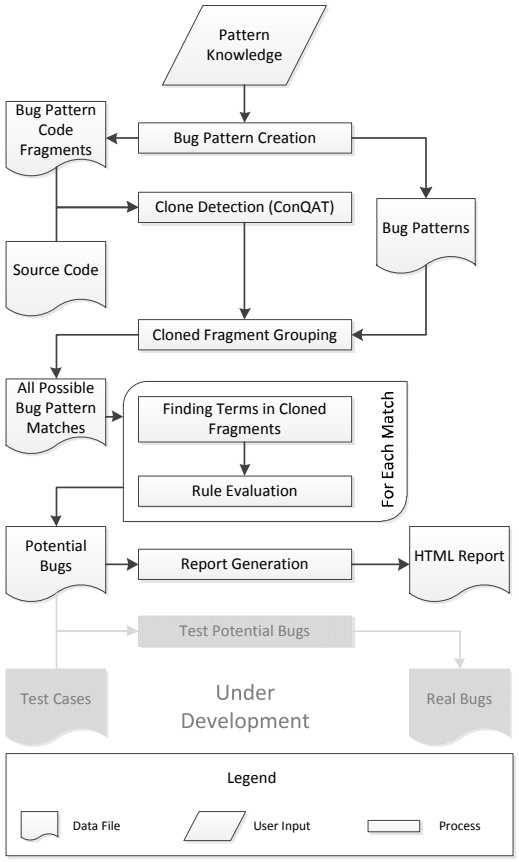

Figure 2: Concurrency Bug Detection Process

5. Proposed Experimental Evaluation

- In order to comprehensively evaluate our active testing research we need to satisfy the following three goals:
  1. Ensure that our specification notation for concurrency bug patterns is expressive enough to handle many different types of concurrency bugs.
  2. Assess our bug detection process and the use of clone detection with finding concurrency bugs.
  3. Evaluate the benefits of using the high-potential bugs to localize testing effort.

6. Conclusions & Future Work

- The search space of concurrent software is very large and the ability to reduce this search space using clone detection, even with the possibility of false positives, can be beneficial in improving testing efficiency.
- Additional work is required to complete our active testing process.
- Experimentation with our tool is required to assess the benefits when compared to existing active testing tools like CalFuzzer [1].

26th IEEE International Conference on Software Maintenance (ICSM)
September 12-18, 2010 ● Timișoara, Romania

Funding provided by: NSERC CRSNG