Exercise 5: Mutation Testing

Gregory Gay
DIT636/DAT560 - February 22, 2024

The Planning System Returns

- Everybody likes meetings.
  - Not true - but we need to book them.
- We don't want to double-book rooms or employees for meetings.
- System to manage schedules and meetings.

Mutate the Meeting Planner

- Create at least four mutants for classes from the MeetingPlanner system.
    - Try to create at least one from each category:
        - invalid (does not compile)
        - valid-but-not-useful (fails for almost any test case)
        - useful (requires specific input or input ranges to detect)
        - equivalent (no test will ever fail)
    - Use different operators for each mutant
        - 1+ from each category in handout.
    - Try mutating different parts of the code.

Assess Your Test Cases

- Run the tests you created in previous exercises. Do they detect the non-equivalent mutants?
    - (Pass on original code, fail for mutated code)
    - If not, create new test cases that will detect them.
    - If equivalent, make sure you understand why the mutant will never be detected.

- If you finish quickly, try this for the CoffeeMaker.

    - (part of Assignment 3)

Example 1

- Valid, but not useful: **constant-for-constant replacement**

```
public boolean isBusy(int month, int day, int start, int end) throws TimeConflictException{
    boolean busy = false; BECOMES
    boolean busy = true;
    checkTimes(month,day,start,end);
    for(Meeting toCheck : occupied.get(month).get(day)){
        if(start >= toCheck.getStartTime() && start <= toCheck.getEndTime()){
            busy=true;
        }else if(end >= toCheck.getStartTime() && end <= toCheck.getEndTime()){
            busy=true;
        }
    }
    return busy;
}
```

```
@Test

public void testIsBusy_NotBusy() {

    // Meeting with no conflict with our dates.

    Meeting noConflict = new Meeting(1,13,1,3);

    Calendar calendar = new Calendar();

    // Add meeting to calendar

    try {

        calendar.addMeeting(noConflict);

        // Enter a time that has no conflict.

        boolean result = calendar.isBusy(1, 13, 14, 16);

        assertFalse(result, "Should cause no conflict");

    } catch(TimeConflictException e) {

        fail("Should not throw exception: " + e.getMessage());

    }

}
```

ANY test where the person is not busy will fail for this mutant!

Example 2

- Useful: Statement Deletion

```
public boolean isBusy(int month, int day, int start, int end) throws TimeConflictException{
      boolean busy = false;
      checkTimes(month,day,start,end);
      for(Meeting toCheck : occupied.get(month).get(day)){
            if(start >= toCheck.getStartTime() && start <= toCheck.getEndTime()){
                  busy=true;
            }else if(end >= toCheck.getStartTime() && end <= toCheck.getEndTime()){
                  busy=true;
            }
      }
      return busy;
}
```

Example 2

- Test passes in invalid date and expects a TimeConflictException to be thrown.

```
@Test

public void testIsBusy_invalid_date() {

    Calendar calendar = new Calendar();

    Throwable exception = assertThrows(

        TimeConflictException.class, () -> {

            boolean result = calendar.isBusy(14, 13, 14, 16);

    });

}
```

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY