

An Industrial Case Study: Using CACC to Test Real-time Embedded Software¹

Jing Guan and Jeff Offutt
ISE Department
George Mason University
Fairfax, VA 22030-4444 USA
{jguan, ofut}@gmu.edu

1. Introduction

Correlated Active Clause Coverage (CACC) is a test criterion whose goal is to test individual clauses within logical expressions [1]. Although considered to be expensive, the FAA requires its close cousin, Multiple Condition Decision Coverage (MCDC), to be used to test safety-critical parts of aeronautics software [3]. This fast abstract presents results from a case study of CACC on a safety-critical, real-time embedded transportation control system. Our study compared CACC testing with functional testing. We found that the CACC test cases detected important faults that were not detected by functional testing, and that would be very difficult to detect by any other testing technique.

2. Background

Generally speaking, *functional testing* derives test cases from the software requirements specification. The functional tests were run on the target hardware without the use of simulation.

CACC testing designs test cases based on the structure of the source code. Each clause must be shown to individually determine the value of the predicate outcome. It is equivalent to the masking form of MCDC [2].

The control system is a very complicated collection of interacting state machines and algorithms that was recently completed and deployed. It interacts with a number of hardware devices, including a battery, generator, and several sensors. It is written in C and consists of 12 files and 90 methods, which collectively contain 70 predicates. 50 predicates had one clause, 17 predicates had two, and 3 predicates had three clauses. These predicates resulted in 134 CACC tests. The software used was a complete, though pre-production version.

3. Experiment and Results

The CACC tests were run on the target hardware in a simulation environment, which provided the software with inputs and collected output messages and results. Each test case was run via a test script that sets the initial conditions of the system, calls the simulation program, launches the control system, and displays the results. Several additional routines were written to allow the tester to choose test cases. These routines allow the tester to start the control system in a particular mode, choose the next test case, and operate the unit in normal mode. The test scripts included everything necessary for the test cases to be easily implemented, run and repeated.

A test engineer performed the functional testing directly on the target hardware. Simulation was not used and the results were checked by hand. Eight faults were found.

The CACC tests were able to find eleven additional faults, plus three of the previous faults. Two faults were related to multi-clause predicates and are very unlikely to have been found by any criterion less stringent than CACC. Two faults were in the software design, one could not be found without simulation, and six required analysis of outputs that were difficult or impossible to observe without simulation.

Two faults were a result of late changes in the software that were not implemented in all locations. The first is outlined here.

In normal operation, the unit goes to different modes based on two inputs—here called *I1* and *I2*. A specific value for variable *X* puts the unit into mode M1, while a value of another variable *Y* puts the unit into mode M2. The following decision encodes this operation:

Decision Block 1:

```
if (X >= 50)
    modeM1();
```

¹ Certain details are omitted from this fast abstract to protect the company's confidentiality.

```
else if (Y >18)
    modeM2();
else if (Y <=18)
    modeM3();
```

In another location in the program, the unit wakes up from sleep mode to normal operation at the same X decision point (50):

Decision Block 2:

```
if ((X < 50) && (Y <=18))
    modeM3();
```

Once the system goes to normal operation, it makes decisions based on Decision Block 1.

After the initial implementation, it was decided to use 45 as the decision point instead of 50, resulting in the following change to Decision Block 1:

```
if (X >= 45)
    modeM1();
```

The fault was that Decision Block 2 was not updated with the change to 45. In the functional test, when 45 was applied to the sleeping unit, Y was also greater than 18, so Y woke up the unit instead of the value of 45 for X . Then when Decision Block 1 was reached, 45 put the unit into mode M1, so the tester erroneously concluded that it seemed as if 45 woke up the unit. This fault is very unlikely to be caught without using a test criterion that forces multiple clauses to be tested independently such as CACC.

Two faults were a direct result of mistakes in the design specification. For example, in operation mode, a digital-to-analog converter is loaded with the value P to adjust the output voltage for battery charging. The software changes the output voltage by increasing a variable P based on the predicate " $((Q \geq 14.7) \ \&\& \ (P < 880))$." The specification said to initialize P to 900 to set the Q to 10.5, and then the value for Q should be increased slowly to 14.7 by decreasing P . When it gets to 14.7, P should be much less than 880, so the condition can be satisfied and the voltage is adjusted. However, when another variable $R < 10.5V$, the Q is immediately set to 14.7. If this happens at the beginning of the process right after P is initialized to 900, the clause $(P < 880)$ will not be satisfied, and then the value for Q will not be decreased. This fault could cause

the unit to keep on increasing above the required limit, resulting in high voltage that would damage the battery. During functional testing, a fully charged battery was used to supply voltage, so the fault could not be detected.

An interesting result is that a number of faults were found that could not have been observed without using simulation. As an example, the software constantly samples a sensor. The variable that stores this number is declared as an `unsigned char`. By using simulation in CACC testing, it was found that the value could be over 255, which could cause it to "roll over" to 0, in which case the software would make a wrong decision based on this incorrect input. Functional testing never caused this condition, so the error was not detected.

4. Conclusions

This fast abstract presents preliminary results from a case study of CACC applied to safety-critical embedded software. Tests to satisfy the CACC criterion coverage found several significant faults that were not found by functional testing. It is unlikely that some of these faults could be detected with weaker test criteria. Automation and simulation was also found to be necessary to find faults. Finally, because of the close connection between the hardware and the software, testing real-time embedded software requires knowledge of both the software and hardware. High-end testing strategies are not often used in practice; this case study gives evidence that they can be useful and effective for safety-critical embedded software.

5. References

- [1] Paul Ammann and Jeff Offutt, *Coverage Criteria for Software Testing*, in preparation.
- [2] John Chilenski and L.A. Richey "Definition for a Masking Form of Modified Condition Decision Coverage (MCDC)", Boeing Technical Report, 1997.
- [3] RTCA-DO-178B, "Software Considerations in Airborne Systems and Equipment Certification," 1992.