Defence Science Journal, Vol. 56, No. 1, January 2006, pp. 13-19

Ensuring Software Quality–Experiences of Testing *Tejas* Airdata Software

Kavitha Rajalakshmi, Y.V. Jeppu, and K. Karunakar

Aeronautical Development Agency, Bangalore–560 017

ABSTRACT

Two major safety-critical elements of the onboard software for the *Tejas* digital flight control computer software are the control laws and the airdata algorithm. The airdata algorithm computes essential parameters like static and dynamic pressures, altitude, speed, angle of attack, etc from the airdata sensor input. These parameters are used by the control laws to stabilise the aircraft and to provide the required uniform handling qualities over the complete flight envelope. The algorithm is provided by the Control Law Design Team and coded by the Software Design Group of Software House, ADA, Bangalore, in Ada language. The Independent Verification and Validation Group is responsible for ensuring that the software is bug-free and certifiable. A non-real time (NRT) test methodology has been developed in-house to stress test the onboard software. This paper gives an overview of the methodology used to carry out the NRT test of the airdata algorithm and some of the testing experiences.

Keywords: Software quality, software testing, best practices, software verification and validation, safety-critical code, *Tejas*, light combat aircraft, quality assurance, digital flight control computer software, airdata algorithm, non-real time test methodology

1. INTRODUCTION

The non-real time (NRT) test methodology developed at the Aeronautical Development Agency, Bangalore, for testing safety-critical software[1] in 1998 has matured over the years[2,3]. A randomised way of generating test cases has been developed to remove some of the drawbacks of manually-generated test cases[4]. The method has been used to test various builds of the digital flight control computer (DFCC) software. The Software Testing Group at the Aeronautical Development Agency (ADA), Bangalore has gained vast experience in testing of the software using NRT test methodology over the years. The software defects, which have passed preliminary unit-level and integration-level tests have been trapped. Some defects have even passed through system- level tests at the Ironbird Facility. This paper consolidates the Software Testing Group's experience in ensuring software quality using NRT test methodology. A few interesting catastrophic defects have been analysed in this paper.

The airdata algorithm is briefly presented to familiarise about the system under test. A description of the NRT test methodology is provided for completeness. A few category 1 (catastrophic) defects have been presented and the nature of the defects has been discussed providing an insight into the power of NRT test methodology in trapping these

Received 04 October 2005

defects. The causes of the defects and the ways these could have been avoided or trapped earlier, are also discussed. Finally, the best practices gleaned from the test activity are listed here.

2. AIRDATA ALGORITHM

Modern day aircraft with fly-by-wire flight control systems require information regarding altitude, speed, angle of attack, and sideslip to ensure stability and good handling during the entire flight envelope[5]. These parameters were measured by pressure probes and conventional instruments in earlier aircraft. Today, the same sensor output are digitised and used with sophisticated algorithm to provide a redundant source of information required for feedback in the safety-critical control laws.

The light combat aircraft, *Tejas* has three pressure probes, a temperature probe, and two angle of attack vanes to measure the required parameters. The pressures were converted to electrical signals and transmitted to the four-channel digital flight control computer (DFCC) by RS422 links. Vane and probe de-icing heaters were also monitored for their health. Failure of heaters could cause icing of the probes and vanes, making their measurements invalid. Since the sensor failures could cause mistrack, so the faulty sensors were voted out. Multiple failures could cause certain events to be set in the control law to facilitate reversionary backup control laws to be brought into action. The airdata algorithm output are also sent for display on the head-up-display and on the get-U-home panel for information to the pilot. This information is not safety-critical and a single working sensor can provide the required information. Figure 1 shows the schematic of the airdata system used in the *Tejas* aircraft.

The airdata algorithm was designed by the National Control Law Design Team (NCT), and was coded by the Software House, in Ada language. The Software Design Group of ADA provides a functionality document, a detailed block diagram, and a functional code for the test case and expected result generation. The Independent Verification and Validation (IV&V) Group and the Software Design Group together tested the safety-critical code using the NRT test methodology.

3. NON-REAL TIME TESTING

The NRT test methodology tests the 4-channel DFCC code in a single-strand mode on a target

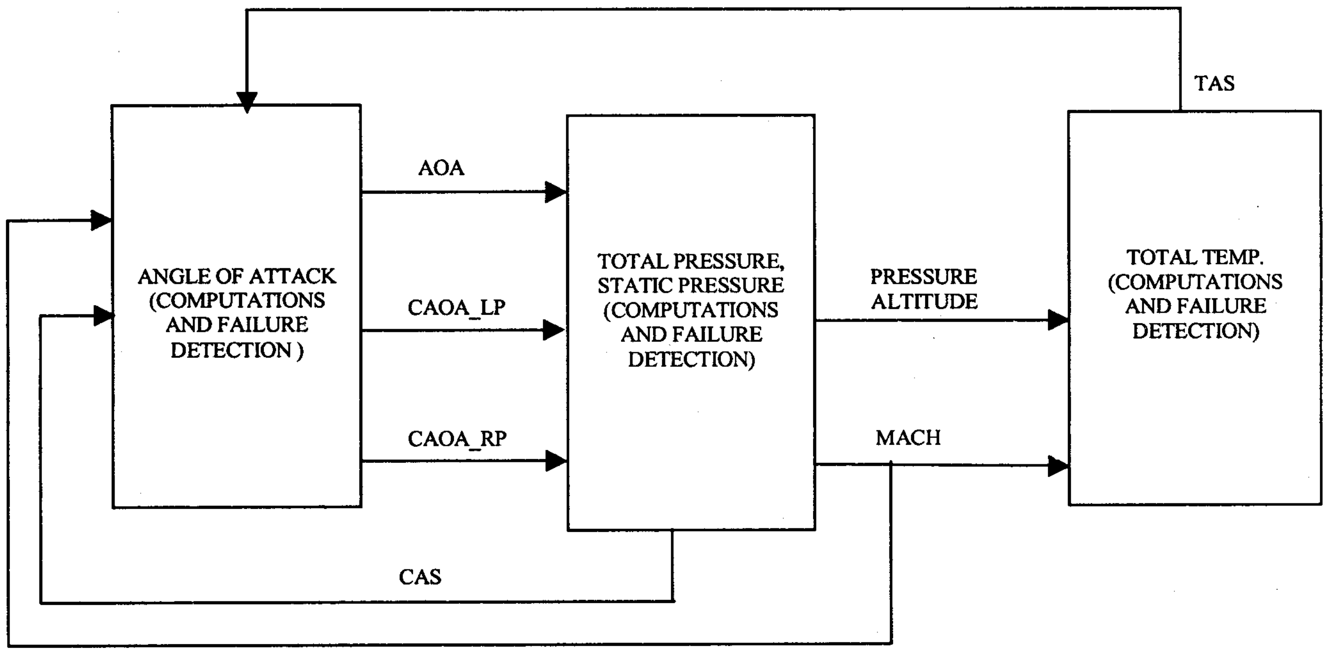
TAS
ANGLE OF ATTACK (COMPUTATIONS AND FAILURE DETECTION)
AOA
CAOA_LP
CAOA_RP
TOTAL PRESSURE, STATIC PRESSURE (COMPUTATIONS AND FAILURE DETECTION)
PRESSURE ALTITUDE
MACH
TOTAL TEMP. (COMPUTATIONS AND FAILURE DETECTION)
CAS

Figure 1. Schematic of airdata system

hardware board. The target hardware board is a single-board computer with the same processor as the 4-channel DFCC. The single-strand code is compiled after clipping and stubbing with the same options as the final software build. The executable is run on the single-board computer and the output compared with the design code. A defect in the software is certain if the error between the two is greater than a specified threshold (0.02 %).

4. SOFTWARE DEFECTS

Various builds of the DFCC codes have been tested since 1998. Several software defects (category 1) have been detected during the tests. Errors (category 1) can cause catastrophic failures of the safety-critical code. These defects have been isolated by debugging and testing the safety-critical code. The Software House has rectified the safety-critical code and cleared the software for the flight. A few of these defects are discussed here. The lessons learned in software quality assurance are given as a moral of the test story.

4.1 Index Exceedance

The airdata algorithm has several look-up tables to compute the various parameters. In the total temperature measurement component, there is a table to compute the correction factor based on the Mach number. If the Mach number is >1.0 and ≤ 2.0, then the correction has to be applied. The correction factor is provided as a look-up table wrt Mach number.

During NRT test methodology stress testing Mach number was varied from –1.0 to 3.0 as a sine waveform and the output monitored. It was found that with the increase in Mach number from 2.0 and above, the execution terminated with range-constraint error. Examination of the safety-critical code indicated that for a Mach number value of 2.0, the computed array index exceeded the defined size of the array.

It should be noted that the 4-channel DFCC code is normally compiled with the options to suppress the run-time check. In this case, the execution would not terminate and would continue with the some junk value assumed for the variable.

This interpolation algorithm design defect should have been trapped at the unit-level testing itself. However, this was not done because of the inadequacy of the unit-level test cases to check the boundary values. Another important issue was, during testing phase, the compiler option, 'suppress run-time checks' was set. So, even if the boundary checks were performed and if the error was within the tolerance bounds specified for the test, then the bug in the safety-critical code would not have been noticed.

Test Moral

Always check for (i) the exact boundary values of the variables, (ii) greater than the boundary value of the variable, and (iii) less than the boundary value of the variable during unit-level testing. Stress testing software is necessary during functional and integration-level tests. Dynamic input like high amplitude sine waves used in the NRT test methodology could cause a system failure and trap the defect. During the testing phase, the run-time checks need not be suppressed as otherwise problems like this may go unnoticed.

* Wrong indexing is a very common mistake made very often–look out for this. Bugs can hide anywhere, but these are always present behind the kitchen sink!

4.2 Incorrect Indexing

The total temperature computation block has two correction factors. One correction is for the de-icing heater error (DHE) and the other for the self-heating error (SHE). Both the correction factors are provided as look-up tables DHE_TAB and SHE_TAB wrt a variable Z. In the algorithm in Ada code, the Z table index was computed for the DHE look-up table. As the Z values used for SHE and DHE were the same, the index computed for DHE was used for SHE also. However, there was a condition of heater failure where DHE index was not computed at all but the SHE correction was required.

During NRT tests, a varying Z value with random toggling of the heater flag was used in the test case. This caused a mismatch between the designer- provided model code and the Ada safety-critical code. Debugging the code brought out the following software design defect:

As the Z index was computed during the DHE computation and used for the SHE interpolation, during heater failure, the index was not computed and SHE computation was erroneous.

Test Moral

An independent model code, as a reference for the expected result, is very essential to benchmark a safety-critical code. Random toggling of events/ flags gives a good way of shaking down a piece of code. Randomness in test signals mimics the natural manner in which the flags are set. Special care should be taken when designing software, especially when an advantage is foreseen in cutting down code and optimising. The solution may not be an ideal one in terms of functionality.

* Stress test with random signals–this is a very cheap and efficient way of testing a code.

4.3 Inaccuracies in Table Compression

The look-up table of the algorithm takes up too much memory space. An optimisation was carried out by packing three numbers together. The Software Design Group of ADA provided an algorithm for this. During NRT testing, it was found that the errors between the packed table data and the tables in the functionality document were more than expected. An analysis of the data-packing algorithm brought out the fact that the Software House had used FIX (extract only the integer value) instead of ROUND (round off to the nearest integer) to compute the packed table. This error, though small, caused an increase in errors in the computed parameters due to the presence of a multiplication factor.

Test Moral

As a tester of the system (not just the software), it pays to check algorithm design also. NRT test methodology provides a digital platform, where such small errors can be trapped. This cannot be done with the actual hardware like A/D converters present, as their noise would mask the test accuracies.

* Broaden the test scope without compromising test schedules.

4.4 Missing Requirements

During flight, it was found that the airdata parameters were a function of the aircraft undercarriage operation. A detailed post-flight analysis was carried out and the algorithm modified to cater for this. This change was reflected in the functionality document released for coding. However, the final code released for testing did not have this correction. The effect of the change was small but necessary. System- level testing cleared the software for flight.

NRT testing revealed the defect in the software. This was due to the dynamic nature of the testing and the very tight error bands involved in the testing.

Test Moral

Changes made to the design should be highlighted and mentioned separately. The Software Design Group of ADA should take special care to see that the design modifications are understood and coded. A meeting with the Coding team and Independent Varification and Validation (IV&V) Group highlighting the changes would have prepared both for the additional testing.

The unit-level test cases also used similar code design and were perhaps made by the same group. Independent check by a separate group is essential to check the software. In places where there is manpower shortage, software coder from another group could be used to check the software by interchanging their work. System-level tests are not meant for stress-testing software. These are useful for demonstration of system performance. It is very essential that system design personnel be involved in testing. This synergy leads to better software quality!

* Involve the system designers and software designers in test activity. The combination works very well.

4.5 Cut-paste Errors

The Software House reported errors in the altitude (a derived parameter) during its software integration tests (SIT), which were very high. The explanation given by them was that the errors were due to the approximations in the look-up table due to packing. The Software Design Group checked this with the maximum errors in the tables due to packing and for various flight conditions. It was found that errors due to packing were much smaller than what was encountered during software integration tests (SIT). Meanwhile, another group, which was looking up at the compiled object code, reported an error but was unable to trace the source of the errors.

NRT testing of the software with large amplitude signals, for stress testing the code and tapping of intermediate variables, could isolate the error to a specific function. A code walkthrough of the function revealed that a two-dimensional interpolation routine was sending the same variable for the X and Y variables instead of sending two separate variables. This is seen in the code example given below. As the other variable was not being used, the optimising compiler had removed that portion of the code as the dead code. This was the error noticed in the object code analysis.

Test Moral

The causes for the defect in the software were attributed to the cut-paste technique used while writing code, and the inadequate unit-level testing.

It is often seen during testing that the errors exist in a visible variable but the cause lies somewhere else. Stress testing is very essential for any safety-critical code. Tapping of intermediate variables gives an excellent view into the working of the software and it is very easy to debug the code. The price paid for tapping out the additional parameters is the slowing down of the test activity. However, automatic testing and logging of the results and errors can overcome this deficit in testing. As experienced, if the test cases are partitioned to test specific blocks, and if all the input to the block and output of the block under test are tapped, it pays rich dividends in terms of time saved in debugging and isolation of the problems.

* If the output is not what was expected, it is most likely to be because of a bug.

4.6 Uninitialised Variable

Initialisation part of the code is the most difficult to test. There are many variables and many values these can take. An uninitialised variable is detected by observing the first frame of data. In case of filters and integrator, an uninitialised variable will show its effect in the subsequent frames. Random tests with initial conditions randomly selected, can check for such situations. The following code shows the initialisation phase and the main code segments. This error was captured as an error visible only in the first frame. The code was later corrected by adding the code segment below.

Test Moral

A set of test cases can be generated for testing only the initialisation part. The test cases generated randomly can be executed for a few frames (say 10 major cycles) to test the initialisation part automatically. Any error in the first few frames is definitely due to an error in the initialisation.

* There is always an error in the zeroth frame!

5. BEST PRACTICES

The experiences gained in the test activity are summarised as best practices in the requirements phase, software coding phase, and the testing phase.

5.1 Requirements Phase

A systematic approach to requirements capture is required. The practice followed some *Tejas* is that the Software Design Group of ADA gives the requirements in the form of a functionality document. It has been observed that the Software House usually misinterprets the requirements. A simulink block diagram and model is provided by the Software Design Group of ADA to the Software House. Problems have been attributed to the Software House not understanding the simulink blocks. Following steps may be taken during the requirements phase:

- An interaction between the Software Design Group and the Software House is a must for proper requirements capture.

- Automated requirements management tools can help remove some of the misunderstanding by giving a text-alone requirement. The simulink blocks can be integrated with the requirements for better understanding.
- It is seen that Software House makes changes in the safety-critical code for optimising performance. These changes should be ratified with the Software Design Group for their effect on functionality. Working of these two groups in isolation is harmful as defects are trapped very late in the project.

5.2 Coding Phase

- Automatic code generators could reduce some of the errors found in the safety-critical software.
- A separate group should carry out extensive unit-level tests, preferably using some of the automated tools for checking dead code.

5.3 Testing Phase

- The success of NRT test methodology can be mainly attributed to interaction between the Independent Verification and Validation Group and the Software Design Group.
- There is, however, minimal interaction with the Software House. Involvement of the Software House in the testing could sort out certain misunderstanding.
- Test case should be generated to test the system in an end-to-end manner.
- Stress testing is essential for any safety-critical software.
- Testing software is not a routine affair. Testers play a very important role in the software development phase. They should be trained in the latest technologies available.
- Management should consider testing as essential and not a hindrance to the project schedule. Proper management of project schedules, with adequate time frames provided for testing, can produce high quality certifiable software.

6. CONCLUSION

Non-real time (NRT) testing has provided a rich experience in ensuring software quality. The concept of a single-strand testing, the digital mode of testing with tight error tolerance bands, and stress testing of the safety-critical code in an end-to-end manner with tapped out intermediate variables, has trapped a number of safety-critical (category-1) errors. A few of these are mentioned with the lesson learnt in the process.

The test morals provide a few best practices for testing and ensuring quality of safety-critical software. These are:

- Common software coding errors will always be present. These may be looked for in particular.
- Stress test with random signals is a very cheap and efficient way of testing safety-criticcal code.
- Be very specific about the scope of the test. It is futile to go on testing the code. But, try and broaden the test scope without compromising test schedules.
- Start the test activity very early. Involve the system designers and software designers in the test activity from the very beginning.
- If the output is not what was being expected, it isvery likely to be because of a bug. There could be other explanations but in 99 per cent of the cases, it will be an error–look for it.
- There is always an error in the zeroth frame. Initialisation errors are very common. Random test cases are very effective in trapping such errors.

A well-trained test team is an asset to the organisation involved in safety-critical software development. The future of software development is formal methods, which purport to do away with software testing completely. But it is essential to remember–to err is human.

* Testing software is inevitable for ensuring quality in any safety-critical software.

ACKNOWLEDGEMENT

The authors are grateful to the Programme Director (Combat Aircraft) and the Director, Aeronautical Development Agency (ADA), Bangalore, for their support and permission to publish this work.

REFERENCES

1. Jeppu, Y.V.; Harichoudary, C.H. & Misra, B.B. Testing of real time control system: A cost effective approach. *In* SAAT 2000, Advances in Aerospace Technologies, Hyderabad, India.
2. Jeppu, Y.V.; Karunakar, K. & Subramanyam, P.S. Flight clearance of safety-critical software using non-real time testing. *In* ATIO, 2002. AIAA Paper No. AIAA-2002-5821.
3. Jeppu, Y.V.; Karunakar, K. & Subramanyam, P.S. Testing safety critical Ada code using non-real time testing. *In* Reliable software technologies, ADA-Europe 2003, edited by Jean-Pierre Rosen and A. Strohmeier. Lecture Notes in Computer Science, 2655. pp. 382-93.
4. Giri, Sukant K.; Mishra, Atit; Jeppu Y.V. & Karunakar, K. A randomised test approach to testing safety critical Ada code. *In* Reliable software technologies, Ada-Europe-2004, edited by Albert Liamosi and Alfred Strohmeier. Lecture Notes in Computer Science, 3063. pp.190-99.
5. Collinson, R.P.G. Introduction to avionics. Microwave Technology Series-11. Chapman & Hall, 1996.

Contributor

Ms K. Kavitha Rajalakshmi has done her BE (Electronics and Communication Engg) and ME (Applied Electronics) from the Government College of Technology, Coimbatore. She is working at the Aeronautical Development Agency (ADA), Bangalore in Independent Validation and Verification Department since 1998. Her main responsibility is verification and validation of real-time embedded system software of safety/mission-critical domains like airdata parameter computation software, redundancy management software, flight test panel, crash data recorder of LCA, etc in which she has used different methodologies in software testing. She has also performed independent validation and verification activities for *Agni* missile developed by RCI, Hyderabad.