Taming the Complexity Beast

Darren Cofer, Ph.D.,
Fellow, Rockwell Collins Advanced Technology Center
Minneapolis, MN

Introduction

Advanced capabilities planned for the next generation of aircraft and the Next Generation Air Transportation System (NextGen) will be based on complex new software. Integrated Modular Avionics (IMA) computing platforms already enable the implementation of more functionality in software and tighter integration of these functions to improve aircraft efficiency. In the future, aircraft may use adaptive and intelligent control algorithms to provide enhanced safety and robustness in the presence of failures and adverse flight conditions. Unmanned aircraft will join the National Airspace System (NAS), incorporating advanced control algorithms that will provide enhanced safety, autonomy, and high-level decision-making functions normally performed by human pilots. NextGen will encompass airborne and ground-based nodes with significant computational elements acting in coordination to maintain a safe and efficient airspace.

However, there are serious barriers to the deployment of these new capabilities. As these systems have grown in complexity, verification of airborne software has become the single most costly development activity (Crum 2004). The verification costs of even more complex NextGen systems in the future may impact safety, not just through an increasing incidence of errors and unforeseen interactions, but by delaying and preventing the deployment of crucial software-based safety functions.

Increasing system complexity therefore poses a threat to the continued safety of manned and unmanned aircraft in the NAS. Testing alone cannot establish strict bounds on all the behaviors that may occur during operation of these software-intensive systems. New approaches to verification based on logic and mathematical analysis are needed to tame the "complexity beast" and support continued innovation in aircraft systems.

This article will briefly describe sources of complexity in modern aircraft software and the limitations of test-based verification methods. The role of software testing in the certification standards for civil aircraft will be described next, as well as how this domain is beginning to embrace new verification approaches based on *formal methods*. The article concludes with several examples of formal methods that have been used to verify complex software.

Complexity and Software Testing

Edsger Dijkstra famously said, "Testing shows the presence, not the absence, of bugs" (Dijkstra 1969). For relatively simple programs, it is possible to exhaustively test all possible inputs and cover all the internal states of the program. However, for most realistic software systems testing can only hope to sample the enormous space of behaviors. Therefore, the fact that a set of test cases does not reveal any errors is not a guarantee that the software is, in fact, error-free. And because of the discrete nature of software, even a large test suite cannot provide the assurance that we might expect in a system with continuous dynamics. This difference is critical for safety-critical software in aircraft.

In addition to the discrete (vs. continuous) nature of software, there are a number of other sources of complexity:

- By any measure, the amount of safety-critical software deployed in commercial and military aircraft is rising exponentially. All other things being equal, this alone increases complexity.
- Flight deck software is not a single, monolithic program, but a collection of asynchronous programs that interact in real-time. In this environment, errors related to transients and race conditions are notoriously difficult to replicate and track down.
- Taken as a whole, an aircraft is a hybrid system, consisting of both discrete (computer hardware and software) and continuous (aerodynamics and the physical environment) elements. While control theory itself is quite mature, the rigorous mathematical analysis of hybrid systems that account for software behavior is still a relatively new field.

For safety-critical systems, the safety assurance process must establish that system reliability is extremely high. In the aerospace domain this has been translated into a probability of failure of less than 10^{-9} per flight hour (SAE International 1996).

It should be obvious that software does not physically fail as hardware does since it is a logical construct not susceptible to wearing out or environmental effects. Any software faults are inherent in its design and present throughout the life of a system. Software is either correct or incorrect with respect to its requirements.

Nevertheless, software systems are embedded in physical environments that are subject to failures and environmental effects (including electromagnetic radiation and high-energy particles.) The physical environment, including pilot commands, may be viewed as a stochastic sequence of inputs to the software. For each input, the program produces either a correct or an incorrect answer. Thus, in a systems context, the software system produces errors in a stochastic manner. To achieve satisfactory statistical significance for failure rates less than 10^{-7} per hour would require over a million years of testing (Butler 1993).

Therefore, measurement of the reliability of software systems through testing alone is a practical impossibility. In practice, the objectives of software testing are to demonstrate a sampled compliance with requirements, and to detect and eliminate as many software design errors as possible. More on this in the next section.

A recent discussion of the foundations of software testing, the limitations of testing, and efforts to improve its effectiveness in detecting errors can be found in (Staats 2011).

Airborne Software Certification

For software in commercial aircraft, software assurance guidance is found in DO-178C, *Software Considerations in Airborne Systems and Equipment Certification*. Certification authorities in North American and Europe have agreed that an applicant (aircraft manufacturer) can use this guidance as a means of compliance with the regulations governing aircraft certification.

The software assurance process makes sure that components are developed to meet their requirements without any unintended functionality. This means that the process will include activities specifically designed to provide evidence that the software does only what its requirements specify and nothing else.

DO-178C defines five levels of software criticality (A – E, with level A being the most critical) with specific objectives, activities, and evidence required for each level. The processes and objectives in the document assume a traditional development process and rely heavily on test-based verification.

Guidance specific to new software technologies is provided in *supplements* which can add, modify, or replace objectives in the core document. New supplements were developed in the areas of object-oriented design, model-based development, and formal methods, as well as an additional document containing new guidance on tool qualification. DO-178C and its associated documents were published in 2011 and accepted by the FAA as a means of compliance in 2013.

DO-178C does not prescribe a specific development process, but instead identifies important activities and design considerations throughout a development process and defines objectives for each of these activities. It assumes a traditional development process that can be decomposed as follows:

- Software Requirements Process. Develops High Level Requirements (HLR) from the output of the system design process.
- Software Design Process. Develops Low Level Requirements (LLR) and Software Architecture from the HLR.
- Software Coding Process. Develops source code from the Software Architecture and the LLR.
- Software Integration Process. Combines executable object code modules with the target hardware for hardware/software integration.

Each of these processes produces or updates a collection of artifacts, culminating in an integrated executable. The results of these development processes are verified through the verification process. The verification process consists of review, analysis, and test activities that must provide evidence of the correctness of the development activities.

In general, verification has two complementary objectives. One objective is to demonstrate that the software satisfies its requirements. The second objective is to demonstrate with a high degree of confidence that errors that could lead to unacceptable failure conditions, as determined by the system safety assessment process, have been removed.

One of the foundational principles of DO-178C is requirements-based testing. This means that the verification activities are centered around explicit demonstration that each requirement has been met. Test cases must be developed for both normal and abnormal input ranges to demonstrate robustness.

A second principle is complete coverage, both of the requirements and of the code that implements them. This means that every requirement and every line of code will be examined in the verification process. Furthermore, several metrics are defined which specify the degree of structural coverage that must be obtained in the verification process, depending on the criticality of the software being verified. The use of structural coverage metrics are

key in that they can help identify missing requirements and unintended functionality.

A third principle is traceability among all of the artifacts produced in the development process. This means that:

- Every requirement must have one or more associated test cases. All testing must trace to a specific requirement.
- Every requirement must be traceable to code that implements it. Every line of code must be traceable to a requirement.
- Every line of code (and, in some cases, every branch and condition in the code) must be exercised by a test case.

Together, these objectives provide evidence that all requirements are correctly implemented and that no unintended function has been implemented. As discussed above, this evidence is not a *guarantee* of correctness. Historically, it has been sufficient to produce highly reliable software for aircraft. Our concern is whether it will continue to be sufficient in the face of increasing software complexity.

Formal Methods in Certification

So if testing is inadequate for verifying complex software systems, what is the alternative? *Formal methods* are mathematical techniques for the specification, development, and verification of software aspects of digital systems. Formal methods are based on formal logic, discrete mathematics, and computer-readable languages. The use of formal methods is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analyses can contribute to establishing the correctness and robustness of software. Formal methods for software analysis can be viewed as the analog of finite element analysis for mechanical structures.

Unlike testing, formal methods can provide a complete assessment of software behavior, limited only by the soundness of the modeling abstractions that are used. A formal analysis is a proof of correctness of the design relative to its requirements. If we can apply formal methods to the verification of software or a software design, we will be able to cope with growing complexity in a way that is impossible for test-based verification.

DO-333, the Formal Methods Supplement to DO-178C, extends the guidance provided in DO-178C and describes how formal methods may be used to satisfy its certification objectives. Several case studies showing examples of how to use different formal verification tools to satisfy various certification objectives are available in (Cofer 2014). DO-333 generally allows the testing described above to be replaced by a comparable formal analysis. However, even when formal methods are used some on-target testing is still required.

One constraint imposed by DO-178C is that requirements must be verifiable, which in the past has meant "testable." This meant that in practice there could be no negative requirements such those related to safety (e.g., "The system can never enter an unsafe state.") However, with the advent of DO-333, such requirements can now be addressed analytically and may be very useful in demonstrating the safety of a complex avionics system.

Complex Components

The wide-spread use of *model-based development* (MBD) tools is facilitating the use of formal methods for verification. MBD refers to the use of domain-specific (often graphical) modeling languages that can be executed in simulation before the actual system is built. The use of such modeling languages allows engineers to create a model of the system, execute it on their desktop, and automatically generate code and test cases. Furthermore, tools are now available to translate these design models into analysis models that can be verified by formal methods tools with the results translated back into the original modeling notation. This process leverages the original modeling effort and allows engineers to work in familiar notations for their domain.

Model checking is a category of formal methods that is particularly well suited to integration in MBD environments. A model checker will consider every possible combination of system input and state, and determine whether or not a specified set of properties is true. If a property is not true, the model checker will produce a counterexample showing how the property can be falsified. Model checkers are highly automated, requiring little to no user interaction, and provide the verification equivalent of exhaustive testing of the model.

In the Certification Technologies for Flight Critical Systems (CerTA FCS) project funded by the U.S. Air Force, we analyzed several software components of an adaptive flight control system for an unmanned aircraft. One system we analyzed was the redundancy manager which implements a triplex voting scheme for fault-tolerant sensor inputs. We performed a head-to-head comparison of verification technologies with two separate teams, one using testing and one using model checking. In evaluating the same set of system requirements, the model checking team discovered 12 errors while the testing team discovered none. Furthermore, the model checking evaluation required 1/3 less time (Whalen 2007).

The case studies in (Cofer 2014) provide examples of three classes of formal methods (model checking, theorem proving, and abstract

interpretation) applied to several different avionics software components. These case studies demonstrate the effectiveness and practicality of using formal methods to verify complex software components in an aircraft certification context.

Complex Systems

As system size and complexity grow, verification demands can easily exceed the capabilities of current formal methods tools. System-level verification can be accomplished using a compositional approach to break down the analysis task into manageable pieces according to the system architecture. Furthermore, many important sources of errors appear at the system architecture level. However, tools for modeling and analyzing system-level properties using formal methods have been quite limited.

Without the means to rigorously model the system architecture, system and safety engineers are unable to effectively communicate and review ever more complex system designs. Without tools to effectively analyze behaviors resulting from the system architecture, most system-level design errors will not be detected until system integration when the cost of correction is far greater and likely to introduce still more errors. As more functions are implemented as asynchronous software components, testing becomes less and less effective at finding race conditions and deadlocks, requiring greater emphasis on analysis. Without a precise specification of the system architecture, analytic techniques can only be applied to hand-crafted models that are unlikely to represent the true system design and that may not be completely trusted by developers.

Our research group is addressing these challenges by developing compositional reasoning methods and tools that will permit the verification of systems that exceed the complexity limits of current approaches. Our approach is based on:

- Modeling the system architecture using standard notations that will be usable by systems and software engineers.
- Developing a sophisticated translation framework that automates the translation of these models for analysis by powerful formal methods verification tools.
- Developing techniques for compositional verification based on the system architecture to divide the verification task into manageable, reusable pieces.

This approach has the potential to significantly reduce verification costs by identifying and correcting system design errors early in the life cycle rather than waiting until system integration. We are validating our approach and our tools on a realistic fault-tolerant flight control system model. The Quad-redundant Flight Control System (QFCS) has been designed by NASA as a suitable control system for its Transport Class Model (TCM) aircraft (Backes 2015).

Our compositional approach is designed to exploit the verification effort and artifacts that are already part of typical software component verification processes. Each component in the system model is annotated with an assume/guarantee contract that includes the requirements (guarantees) and environmental constraints (assumptions) that were specified and verified as part of its development process. We then reason about the system-level behavior based on the interaction of the component contracts. By partitioning the verification effort into proofs about each subsystem within the architecture, the analysis will scale to handle large system designs. Additionally, the approach naturally supports an architecture-based notion of requirements refinement: the properties of components necessary to prove a system-level property in effect define the requirements for those components.

There were two objectives in using this verification approach. The first was to reuse the verification already performed on components. The second was to enable distributed, parallel development of components via virtual integration.

In this process, we specify formal component-level requirements, demonstrate that they are sufficient to prove system guarantees, and then use these requirements as specifications for suppliers. If the suppliers' implementations meet these specifications, we have a great deal of confidence that the integrated system will work properly.

We have chosen the Architecture Analysis and Design Language (AADL) as our system architecture modeling language (Fieler 2012). AADL was designed for embedded, real-time, distributed systems and so is a good fit for our domain. It provides the constructs needed to model embedded systems such as threads, processes, processors, buses, and memory. It is sufficiently formal for our purposes, and is extensible through the use of language annexes that can initiate calls to separately developed analysis tools.

We have implemented our compositional reasoning methodology in a tool called AGREE: Assume-Guarantee Reasoning Environment. AGREE is implemented as an Eclipse plugin and is designed to work with the open source OSATE AADL tool developed by the Software Engineering Institute. AGREE is able to check the correctness of behavioral properties defined by the composition of component contracts, check component contracts for inconsistencies, and determine whether a component contract has any possible realization. AGREE makes use of the AADL annex mechanism to annotate models with contracts corresponding to formal

assumptions and guarantees about their behaviors. AGREE is open source software and is available at http://github.com/smaccm.

What Next?

Beyond the challenges of complexity in software components and system designs, there are new challenges related to unmanned aircraft. Unmanned aircraft having the advanced capabilities necessary to operate safely and autonomously in the NAS will likely be based upon software including adaptive control (AC) and artificial intelligence (AI) algorithms.

The current civil aviation certification process is based on the idea that the correct behavior of a system must be completely specified and verified prior to operation. The fact that adaptive systems change their behavior at run-time is contrary to this idea in many ways. In general, many AI methods have unique characteristics that do not fit naturally within context of existing certification guidelines. This is due to the fact that the certification policies, conceived decades ago and still in use today, were not written with the needs and capabilities of AI in mind (Harrison 1994).

While systems based on artificial intelligence and adaptive algorithms can be found in military and space flight applications, they have had only limited use in civil airspace due to the constraints and assumptions of traditional safety assurance methods. These barriers may delay or prevent the deployment of some unmanned aircraft in the NAS.

An overview of these systems and the challenges they present can be found in (Bhattacharyya 2015). Certification challenges for these systems may include:

- The difficulty associated with specifying and verifying the behavior of AC and AI algorithms in software
- The use of non-traditional programming languages and the difficulty of measuring structural coverage in these languages
- The inclusion of non-deterministic behaviors
- The additional complexity of AC and AI systems compared with traditional systems
- The fact that certification authorities are generally unfamiliar with these systems

We have explored several mitigation strategies to address these challenges.

In our experience, there can be an expertise gap between developers and regulators when it comes to adopting new technologies. In fact, the commercial aviation industry is itself very conservative and (for good reason) usually reluctant to switch to the latest technology. However, we are convinced that for some adaptive algorithms this reluctance is unwarranted. Some approaches to adaptive control have been proven to be dependable and predictable in flight tests, and there seem to be no *actual* barriers to their certification. In this case, no changes to the certification process are required.

Current standards assume static behavior specifications for aircraft functions. It may be possible to relax this assumption and other constraints in a principled way. The goal here would be to modify our existing standards in a way that retains the underlying safety principles, but also permits a more dynamic software structure.

Certification approaches based on the development of a *safety case* for the aircraft (including its adaptive components) would in principle provide more flexibility to use advanced algorithms, demonstrating the safety of the adaptive algorithm by using the most appropriate evidence, while not sacrificing safety. However, there is much work to be done before applicants would have sufficient expertise to produce an accurate and trustworthy safety case, and regulators would be prepared to evaluate one (Holloway 2014).

Current test-based verification processes will never be sufficient to assess the behavior of adaptive systems. Factors such as software size, complexity, unconventional artifacts, probabilistic computations, and large state spaces have been discussed as reasons for the difficulty of testing. Testing will have to be replaced or augmented by analysis based on formal methods or other mathematical techniques from the control theory or computer science domains.

There may be architectural approaches that could mitigate certification barriers for adaptive systems. Suppose that we are to certify an adaptive function that provides some advanced capability related to improved performance or recovery from a failure or upset, but we are unable to verify the behavior of the function with the required level of assurance. It may be possible to bound the behavior of the adaptive function by relying three smaller, high-assurance functions: a system status monitor, a simpler backup for the adaptive function, and a switching function. During normal operation, outputs from the adaptive function are used by the rest of the system. If the monitor detects that the adaptive function is not behaving correctly then the system will switch to using outputs from the simpler backup function. The key idea is to be able to treat the adaptive system differently based on when it executes (e.g., during different phases of flight).

Conclusion

Advances in aircraft performance and safety will be based on software-intensive systems with frightening levels of complexity. However, we need not fear this complexity if we are willing to tame it using appropriate tools. Moving from test-based

verification to analysis-based verification will be critical to maintaining the safety record that the civil aerospace industry provides today.

DARREN COFER, Ph.D., is a Fellow at the Rockwell Collins Advanced Technology Center. He earned his Ph.D. in electrical and computer engineering from The University of Texas at Austin.

His principal area of expertise is developing and applying advanced analysis methods and tools for verification and certification of high-integrity systems. His background includes work with formal methods for system and software analysis, the design of real-time embedded systems for safety-critical applications, and the development of nuclear propulsion systems in the U.S. Navy.

He has served as principal investigator on government-sponsored research programs with NASA, NSA, AFRL, and DARPA, developing and using formal methods for verification of safety and security properties. He is currently the principal investigator for the air vehicle team in DARPA's High Assurance Cyber Military Systems project, focusing on formal proof of security properties for unmanned air vehicles.

Dr. Cofer recently served on RTCA committee SC-205 developing new certification guidance for airborne software (DO-178C) and was one of the developers of the Formal Methods Supplement (DO-333). He is a member of the RTCA Forum for Aeronautical Software, the Aerospace Control and Guidance Systems Committee (ACGSC), and a senior member of the IEEE.

Email: darren.cofer@rockwellcollins.com

References

Crum, V., J. Buffington, G. Tallant, B. Krogh, C. Plaisted, R. Prasanth, P. Bose, T. Johnson. 2004. Validation and verification of intelligent and adaptive control systems. In Proceedings Aerospace Conference, 2004, IEEE.

SAE International. 1996. ARP4761, Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment.

Feiler, P. H., D. P. Gluch. 2012. Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. 1st edn. Addison-Wesley Professional.

Dijkstra, E., J.N. Buxton and B. Randell, eds., 1969. Software Engineering Techniques, April 1970, p. 16. Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27–31 October 1969.

Butler, R., G. Finelli. 1993. The infeasibility of quantifying the reliability of life-critical real-time software. IEEE Transactions on Software Engineering, 19 (1).

Staats, M., M. Whalen, M. Heimdahl. 2011. Programs, Tests, and Oracles: The Foundations of Testing Revisited. In Proceedings International Conference on Software Engineering, May 21–28, 2011, Honolulu, HI, USA.

Cofer, D., S. Miller. 2014. Formal Methods Case Studies for DO-333. NASA Contractor Report NASA/CR-2014-218244.

Whalen, M., D. Cofer, S. Miller, B. Krogh, W. Storm. 2007. Integration of formal analysis into a model-based software development process. In Proceedings Formal Methods for Industrial Critical Systems, Berlin, Germany, July 2007.

Backes, J., D. Cofer, S. Miller, M. Whalen, 2015. Requirements Analysis of a Quad-Redundant Flight Control System. In Proceedings NASA Formal Methods Symposium, Pasadena, CA. May 2015.

Harrison, L. P. Saunders, J. Janowitz, 1994. Artificial Intelligence with Applications for Aircraft, FAA Technical Center Report DOT/FAA/CT-94/41.

Bhattacharyya, S., D. Cofer, D. Musliner, J. Mueller, and E. Engstrom 2015. Certification Considerations for Adaptive Systems. In Proceedings International Conference on Unmanned Aircraft Systems. Denver CO.

Holloway, C. M. 2012. Towards Understanding the DO-178C/ ED-12C Assurance Case. In Proceedings 7th International IET System Safety Conference, Incorporating the Cyber Security Conference, pp. 15–18, October 2012.