Substantially revised version; original appears in *Proceedings of the Ninth ACM International Conference On Embedded Software (EMSOFT)*, pp. 211–218, Taipei, Taiwan, October 2011.

New Challenges In Certification For Aircraft Software

John Rushby*
Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park CA 94025 USA
rushby@csl.sri.com

ABSTRACT

We outline the current approach to certification of aircraft software, and the rôle of the DO-178B guidelines. We consider evidence for its effectiveness and discuss possible explanations for this. We then describe how changes in aircraft systems and in the air traffic system pose new challenges for certification, chiefly by increasing the extent of interaction and integration.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software Verification

General Terms

Verification

Keywords

Certification, DO-178B, Formal Methods

1. CURRENT PRACTICE

Safety certification assures society at large that deployment of a given system does not pose an unacceptable risk of harm. There are several ways of organizing and conducting certification, but all are conceptually based on scrutiny of an *argument* that certain *claims* about safety are justified by *evidence* about the system. Evidence may concern the system or "product" itself (e.g., tests, formal verification, etc.) or the process of its construction (e.g., qualification of developers, adherence to coding standards, etc.). The argument must consider all possible circumstances of the system's operation, including those where faults afflict its own components, or its environment behaves in undesirable ways, and must demonstrate that the system's design maintains safety in the presence of these *hazards*, and that the design is implemented correctly. Development of the argument and evidence for safety interacts with system design, so these proceed iteratively. For example, one hazard to commercial aircraft is fire in the cargo hold; one way to mitigate this hazard is provide an automatic subsystem for detecting and suppressing fires, but then we need to consider hazards to the reliable operation of this subsystem, and also the new system-level hazards that this additional subsystem might introduce.

Although this Claims-Argument-Evidence (CAE) structure provides an intellectual framework for understanding all certification, different industries, nations, and regulatory bodies organize the actual practice of certification differently. One, relatively new, approach requires the "applicant" to develop a *safety case* [8, 18] that makes explicit the claims, evidence, and argument for the safety of the system; the general content or form of these elements may be specified by regulation or guidelines, but the applicant generally has freedom to select or invent the methods to be used within those constraints.

The safety case approach to certification may be contrasted with the *standards-based* approach, where the applicant is recommended or required to follow certain guidelines and standards. These generally specify the assurance processes that should be used, the intermediate artifacts to be produced (requirements, specifications, test plans etc.), the kinds of reviews, tests, and analyses that should be performed, and the documentation required to tie all these together. Standards may be *prescriptive*, meaning that they mandate or strongly recommend particular methods and processes for development and assurance, or based on *objectives*, meaning that they specify what has to be achieved but not how to do it. In both cases, the products and documents generated by following a standard may be considered to constitute evidence when viewed from the perspective provided by the CAE framework; the claims in these cases are generally established by regulation, but where is the argument?

Guidelines and standards emerge from a social process within professional and regulatory bodies, and we can think of that social process as constructing a generic safety case for the class of systems considered; development and examination of the safety argument presumably informs the internal debate that decides what evidence the standard should require, but it is generally not formulated explicitly, nor recorded.

*Supported by NSF grant CNS-0720908 and by NASA contract NNA10DE79C; the revision was supported by NASA contract NNA10DE79C. The content is solely the responsibility of the author and does not necessarily represent the official views of NSF or NASA.

Viewed from the perspective of the CAE framework, safety cases and standards-based certification can be seen as fundamentally similar, but the two approaches do have their own advantages and disadvantages. Standards-based approaches generally incorporate much accumulated experience and community wisdom, and they establish a solid "floor" so that systems developed and assured according to their recommendations are very likely to be adequately safe. On the other hand, standards tend to be slow-moving and conservative, and can be a barrier to innovation in both system design and in methods for assurance. Furthermore, a generic standard may be ill-suited to the specifics of a given system—so that its application may be excessively onerous in some areas, yet provide insufficient scrutiny in others. Because the safety argument is not explicit, the latter deficiency can go unrecognized and the system may be certified inappropriately—for the only requirement is that the evidence should satisfy the standard.

An explicit safety case can be customized very precisely for the specific characteristics of the system concerned, and therefore has the potential to provide stronger assurance for safety than a standards-based approach, and at lower cost (by eliminating unnecessary effort). Furthermore, safety cases can be more agile, allowing greater innovation than standards-based methods. However, there is concern about the trustworthiness of certification based on safety cases, particularly when some of the elements are novel [44] (e.g., an independent review of the crash of a Nimrod military aircraft in 2006 found that its safety case was worthless [17]). The social process that generates standards, and the infrastructure and skill base that develop around them, may provide stronger collective support than is available for a solitary safety case.

Certification of aircraft software is largely standards-based; in fact, it is a quintessential example of the objectives-based variant of this approach to certification and reveals many of its benefits and difficulties. For reasons that will be discussed later, aircraft computer systems and their software are not certified separately, but only as part of a complete aircraft or engine. When a new (or modified) aircraft type is submitted for certification, the certification authority (in the United States, this is the FAA), in consultation with the applicant (i.e., the airframe manufacturer), establishes the *certification basis*, which defines the applicable regulations together with any special conditions that are to be imposed. The applicant then proposes a *means of compliance* that defines how development of the aircraft and its systems will satisfy the certification basis.

Computer systems and software are employed on aircraft to perform specific *functions*, such as primary flight control, autopilot, fuel management, navigation, and so on, and the aircraft-level safety and hazard analysis must consider the possible failure of these functions. Failure includes *mal*function and *unintended* function, as well as *loss* of function; the first two are often more serious than the third. Guidelines for this safety assessment process are provided by Aerospace Recommended Practice (ARP) 4761 [40]. Based on this assessment, the aircraft and system design will be refined to eliminate or mitigate (i.e., reduce the frequency of occurrence, or severity of the consequences) of the various hazards: for example, it may be decided that parts of some function should be fault tolerant, or that a backup system should be provided. Guidelines for this development process are provided by ARP 4754A [39].

The top-level safety claim against which certification is performed would probably be expressed by the layman as "no aircraft should suffer an accident due to a design flaw." However, there are many bad things that can happen to an aircraft that fall short of an immediate accident (e.g., loss of cabin pressure), and the layman would probably agree that we should provide assurance that such things are very rare, but that it might be unreasonable to attempt to eliminate them altogether. This is precisely the approach taken in aircraft certification: the Federal Aviation Regulations (FAR) Part 25.1309[1] identifies five *failure condition categories* from "catastrophic" through "hazardous/severe-major," "major," and "minor" to "no effect." Catastrophic failure conditions are "those which would prevent continued safe flight and landing," while severe-major failure conditions can produce "a large reduction in safety margins or functional capabilities, higher workload or physical distress such that the crew could not be relied on to perform its tasks accurately or completely." There must be an "inverse relationship between the probability and the severity of each failure condition"; in particular, catastrophic failure conditions must be "extremely improbable" while hazardous/severe-major must be "improbable" (recently the preferred term has changed to "extremely remote"). Furthermore, no single failure must be able to produce a catastrophic failure condition. The European Aviation Safety Agency (EASA) Certification Specifications CS 25 are largely harmonized with FAR 25[2] but the most recent version defines a catastrophic failure condition as one that "would result in multiple fatalities, usually with the loss of the airplane." Neither FAR 25.1309 nor CS 25.1309 define "extremely improbable" and related terms; these are explicated in FAA Advisory Circular (AC) 25.1309 and EASA Acceptable Means of Compliance (AMC) 25.1309. These state, for example, that "extremely improbable" means "so unlikely that they are not anticipated to occur during the entire operational life of all airplanes of one type," while "extremely remote" means "not anticipated to occur to each airplane during its total life, but which may occur a few times when considering the total operational life of all aeroplanes of the type."

AC 25.1309 further states that "when using quantitative analyses... numerical probabilities... on the order of 10^{-9} per flight-hour may be used... as aids to engineering judgment... to... help determine compliance" with the requirement for extremely improbable failure conditions. An explanation for this figure can be derived as follows [22, page 37]: suppose there are 100 aircraft of the type, each flying 3,000 hours per year over a lifetime of 33 years (thereby accumulating about 10^7 flight-hours) and that there are 10 systems on board, each with 10 potentially catastrophic failure conditions; then the "budget" for each is about 10^{-9} per hour if such a condition is not expected to occur in the

[1]FAR Part 25 are the regulations for "Large Aeroplanes"; FAR 25.1309 identifies a section within those regulations. The regulations are terse; interpretation and description of "acceptable means of compliance" are generally issued as Advisory Circulars of the FAA; AC 25.1309 is the advisory circular corresponding to FAA 25.1309.

[2]EASA issues the regulations CS 25 and acceptable means of compliance AMC 25 as separate books within a single document [13]. CS 25.1309 and AMC 25.1309 are the EASA equivalents of FAR 25.1309 and AC 25.1309, respectively.

entire operational life of all airplanes of the type. An alternative explanation is given in Section 6a of AMC 25.1309 (see also [22, Chapter 3]): the historical record for the previous (pre-software-intensive) generation of aircraft showed a serious accident rate of approximately 1 per million hours of flight, with about 10% due to systems failure; the same assumption as before about the number of potentially catastrophic failure conditions then indicates each should have a failure probability less than 10^{-9} per hour if the overall level of safety is to be maintained.

Even though recent aircraft types have production runs in the thousands, much higher utilization, and longer service lifetimes than assumed in these calculations, and also have a better safety record, AMC 25.1309 states that a probability of 10^{-9} per hour has "become commonly accepted as an aid to engineering judgement" for the "extremely improbable" requirement for catastrophic failure conditions. The corresponding probabilities for hazardous/severe-major ("extremely remote") and severe ("remote") failure conditions are 10^{-7} and 10^{-5} per hour, respectively.

The safety analysis and development processes of ARPs 4761 and 4754A should result in an aircraft design that minimizes the number and severity of its failure conditions; these processes iterate down through systems and subsystems but at some point we reach components whose internal design is no longer analyzed and refined for safety; instead, we establish requirements for these components and demand that their implementation is *correct* with respect to these requirements. This is how airborne software is treated: the current guidelines DO-178B [28] describe various objectives for documenting and analyzing software that are all focused on ensuring correctness of the executable code (see [1] for an overview); there are similar guidelines for complex designs implemented in hardware [29]. FAA Advisory Circular 20-115B states that an applicant "may use the considerations outlined in DO-178B as a means, but not the only means, to secure FAA approval of the digital computer software" [14].

DO-178B identifies 5 different Design Assurance Levels (DALs) ranging from Level A (the highest) down through Levels B, C, and D to E. Level A is for software whose failure could lead to a catastrophic failure condition, Level B for severe major and so on. DO-178B does not specify how software development and assurance should be performed, but it does specify that these should include certain activities, such as reviews and testing, should produce certain documents, such as plans for various aspects of development and assurance, descriptions of requirements and designs and so on, and that there must be strong configuration management that includes traceability from requirements to code and tests. In all, DO-178B describes 66 objectives of this kind in detail and requires that all of them must be applied to Level A software, 65 of them to Level B, 57 to Level C, and 28 to Level D. Furthermore, at each level, it requires that some of the objectives are performed "with independence" from the development team.

There is a conundrum here: the various failure conditions are associated with tolerable rates of occurrence (10^{-9} per hour, 10^{-7} per hour, and so on) but the assurance objectives associated with the corresponding DALs are all about correctness, and we just do more of them for the higher levels; so how does more evidence of correctness provide assurance for lower rates of failure?

We examine this question in the next section, together with the related questions of whether DO-178B works and, if so, how and why.

1.1 Does It Work? And Why?

Modern aircraft and their software are extraordinarily safe (at least, when flown by airlines and in airspace operated to the standards of North America and Western Europe): no crash in passenger service has been ascribed to software error—although there have been lesser accidents and incidents, which are described below. Furthermore, the most significant recent improvement in aircraft safety has been due to the installation of "Enhanced Ground Proximity Warning Systems" (EGPWS), which have largely eliminated "Controlled Flight Into Terrain" (CFIT) accidents (previously responsible for half of all aviation fatalities), where disoriented pilots fly a perfectly good aircraft into the ground (or a mountain); EGPWS is only made possible by software.

Thus, the historical record suggests that the standards and objectives-based approach to certification employed by DO-178B is effective; why might this be? One reason is surely that the aircraft industry and its regulatory framework are scrupulous about learning from experience: all accidents and incidents are investigated and their reports are models of impartial and conscientious search for underlying causes, and these lessons inform future standards, guidelines, and certifications (recently, however, these benefits are threatened by a move toward criminalization in some jurisdictions [10]). Another is that all passenger aircraft are fundamentally very similar, with changes and innovations occurring in fairly discrete steps (as aircraft "generations") spaced 10 or 20 years apart: hence, the one-size-fits-all character of standards seems well-suited to aircraft (whereas as it might not be for medical devices, where there is a wide range of different kinds of device). Also, the relatively slow rate of change and conservatism of the industry allows the rather ponderous, consensus-driven process for updating standards to keep pace: the original DO-178 was issued in 1982 and updated to DO-178A in 1985; DO-178B was issued in 1992, and DO-178C in 2012. And the guidelines are developed by consortia that include all stakeholders and interested parties (by law, the meetings are open to the public) so that a wide range of knowledge and experience is available.

Separate from the content of standards and guidelines for aircraft software is the matter of their application and oversight. Aircraft safety and certification is assisted by factors that may not be present in all industries. First, there are relatively few companies that develop aircraft systems, and most of these have long experience and a history and culture of safety that underpins their work, so that they may be expected to fully embrace the practice of a standard, rather than merely follow the "letter of the law." Secondly, the system integrators (e.g., Boeing and Airbus) are closely engaged with their system suppliers, and this minimizes the danger that safety issues will "fall through the cracks" between one system and another. Finally, checking that a rather onerous guideline such as DO-178B is fully applied requires a vast amount of regulatory oversight; the European certification agencies charge applicants a fee for their services, but the FAA does not and instead uses "Designated Engineering Representatives" (DERs) to perform much of the oversight. DERs are approved by the FAA but are employed by the companies concerned. Superficially, this may seem like a case of the "fox guarding the henhouse," but it can be argued that DERs can have more intimate knowledge of their specific system than external reviewers, and they allow more manpower to be deployed in oversight [11]. DERs are involved throughout the software development process and conduct structured *Stage of Involvement* (SOI) reviews at strategic points to ensure ongoing compliance with DO-178B.

So we may conclude that the standards-based approach to certification employed with DO-178B does seem to be effective, and that this is probably because its prescriptions are based on experience and are sound, and because they are executed diligently and monitored conscientiously—but also perhaps because of factors outside the standards relating to safety culture, experience, and conservatism.

Methods of software development and assurance evolve much more rapidly than aircraft systems. Thus, while those parts of DO-178B that are mostly focused on the aircraft application (requirements, specifications etc.) retain their currency, those parts concerned directly with software development and assurance (programming, verification, and testing) are challenged by new developments in software engineering. For example, model-based development and autocoding, object-oriented languages, and various kinds of formal methods all offer new opportunities and challenges in the development and assurance of aircraft software. Most of these are addressed in DO-178C, but they point to a general difficulty in the standards-based approach to certification: how can new techniques be accommodated within fixed guidelines?

DO-178B does allow an applicant to propose an "alternative method of compliance" for some objectives, provided it can be shown how their new method satisfies the "intent" of the objectives. The difficulty is that the intent of most objectives is not formulated explicitly. For example, one of the objectives at Level A is testing to a criterion called Modified Condition/Decision Coverage (MC/DC). This is a moderately demanding branch coverage criterion that must be achieved on the code, using tests derived from the requirements. A plausible interpretation of the intent of this objective is that it is to ensure thorough testing of the executable code. However, there are two other intents that can be served by MC/DC testing. First, since the tests must be generated from requirements but achieve high branch coverage on the code, the requirements must be sufficiently detailed that they have essentially the same branching structure as the code: thus, this intent is to ensure very detailed requirements. Second, generating tests from the requirements but measuring coverage on the code will reveal unreachable (i.e., "dead") code; thus, this intent is to ensure absence of unreachable code (because experience has shown it might not be truly unreachable—e.g., due to behavior unanticipated by the requirements—and may then produce unintended function). An alternative method of compliance for the MC/DC objective that satisfies only the "thorough testing" intent but misses the other two might vitiate some of the effectiveness of DO-178B. Thus, a reasonable enhancement to guidelines such as DO-178B would be to include documentation of the intent of each objective. In fact, we could go further than this, for the intent of each objective merely articulates the local part of what should be a coherent overall argument: thus, the really desirable enhancement is for standards-based methods of assurance to supply a full argument that the evidence required by the standard does ensure satisfaction of explicitly stated safety goals.

Part of the argument that should be supplied for correctness-based software guidelines such as DO-178B is a resolution of the conundrum mentioned earlier: how more assurance of correctness, as required by the higher DALs of DO-178B, renders software more suitable for applications that require lower likelihood of failure (i.e., where failure conditions are more serious). This is a significant topic that has received scant attention.

An insightful and original treatment for the conundrum was introduced over a decade ago by Bertolini and Strigini [7] and by Littlewood [20]. The idea is that the top-level claim made for safety-critical software is not that it is reliable, nor that it is correct, but that it is *perfect*. Perfection means that the software will never suffer a safety failure no matter how much operational exposure it receives; it differs from correctness in that correctness is assessed relative to requirements, while perfection is relative to whatever the requirements "should have been" (i.e., to the *right* requirements), because that is how failure is assessed. We will see later that most software errors are due to incorrect requirements; thus, whereas correctness is relative to the high-level software requirements developed according to DO-178B (e.g., exactly how a fuel management system should pump fuel around the various tanks), perfection is relative to the properties considered in the system safety analysis developed according to ARP 4754A for the function implemented by the software (e.g., structural and balance issues concerning the distribution of fuel, and the need to maintain a supply of fuel to the engines).

Now, perfection is a strong claim and we may refuse to accept that software that has been assured to DO-178B Level A is perfect—but we may be willing to concede that it is *possibly* perfect. And we may further be persuaded that its possibility of perfection is greater than software that has been assured only to Level B. This suggests we could attach a (subjective) *probability* to the possibility of perfection.

Probability of perfection is attractive because it relates more naturally than probability of failure to the correctness-based assurance processes used for software. But probability of (im)perfection can also be used to estimate probability of failure; the following sketch of the argument is from [36]. For simplicity, we assume a demand-based system, and consider probability of failure on demand; then, by the formula for total probability

$$\begin{aligned} &P(\text{s/w fails [on a randomly selected demand]}) \qquad (1)\\ &\quad = \; P(\text{s/w fails} \mid \text{s/w perfect}) \times P(\text{s/w perfect})\\ &\qquad + P(\text{s/w fails} \mid \text{s/w imperfect}) \times P(\text{s/w imperfect}). \end{aligned}$$

The first term in this sum is zero, because the software does not fail if it is perfect. Hence, if P_{np} denotes the probability that the software is imperfect, and $P_{f|np}$ the probability that it fails, given that it is imperfect, we have

$$P(\text{software fails}) \leq P_{np} \times P_{f|np}. \qquad (2)$$

(I am cutting a lot of corners here: the full treatment must distinguish aleatoric from epistemic assessment, must justify that beliefs about the two parameters can be separated, and must deal with rates of failure rather than probability of failure on demand; see [21].)

Different industries make different assessments about the parameters to (2). Nuclear protection, for example, assumes the software *is* imperfect, so it sets P_{np} to 1 and undertakes extensive random testing to substantiate (typically) $P_{f|np} < 10^{-3}$. If nuclear regulators were prepared to accept that modest amounts of software assurance could deliver $P_{np} < 10^{-1}$, then assurance for the same probability of failure could be achieved with the much less costly testing required to validate merely $P_{f|np} < 10^{-2}$. Dually, aircraft certification assumes the software *will* fail if it is imperfect, and so sets $P_{f|np} = 1$. The whole burden for assurance then rests on the value assessed for P_{np}. If we suppose that the operational exposure of modern aircraft software and the absence of software-induced crashes substantiates a failure rate below 10^{-9} for Level A software, then this implies that DO-178B delivers assurance for a probability of imperfection of the same order.

I am skeptical of this conclusion, for although there have been no crashes attributed to software, there is one accident (so classified because there were serious injuries) [3] and several incidents where software was involved (e.g., [2, 41, 43]; Daniels [9] provides a longer list culled from [19]). The operational exposure of the software involved in these incidents must be far short of 10^9 hours, so the overall failure rate of recent software may be of the order of 10^{-7} per hour, or worse. Then, although developers do not take credit for it in DO-178B, aircraft software is subjected to massive amounts of system and all-up testing. It is plausible that this is sufficient implicitly to establish $P_{f|np} < 10^{-3}$. Substituting these values in (2), it is therefore possible that DO-178B Level A actually delivers only about $P_{np} < 10^{-4}$.

Thus, although the safety record of airborne software is good, it is worth examining whether some aspects of its development and assurance processes can be improved.

1.2 What Goes Wrong? What Might Fix It?

The software flaws described in some of the incident reports cited above are egregious (e.g., see [21], which provides a brief description of the flaws fully reported in [43]) and one wonders how they could have passed DO-178B. One possibility is that DO-178B alone is not a strong guarantee, and the generally good safety record of aircraft software is partly due to other factors mentioned earlier, such as the long experience and safety culture of the companies concerned, the oversight of the system integrators, and so on. If this is so, then recent industry trends raise concern: there has been massive outsourcing of software development and assurance to companies in the developing world, where there is no tradition of a safety culture and whose DERs are external consultants rather than company employees (and may therefore lack *tacit knowledge* about the software and system concerned [11]), and the system integrators do not monitor their subcontractors as closely as before.

Another possibility is that DO-178B is effective in some areas and less so in others; hence, is important to try to understand when and *why* DO-178B works, and what is the contribution of its various objectives and its organizational context. Academic study of these questions is difficult because the companies concerned do not publish their internal data: it is only when a failure provokes an incident that information can be gleaned from the ensuing report.

One item that can be learned from those reports and that is supported by anecdotal evidence is that essentially all flight software failures (including [43]) are due to improper software requirements, and none are due to programming errors. Thus, it seems that a vulnerability may lie in the gap between the system requirements developed through ARP 4754A and the high-level software requirements developed through DO-178B, even though the DO-178B objectives demand evidence that the software requirements comply with and are traceable to the system requirements.

One approach to reducing this vulnerability could be to drive safety analysis (rather than correctness) down into the top levels of software development. Rockwell Collins report value in applying model checking to software requirements to check, for example, that they satisfy certain safety properties [23]. I believe there could be value also in using formal methods as part of the safety analysis of the system requirements themselves: as systems become more complex, so their requirements take on more of the characteristics of software, with huge numbers of cases to consider, and it makes sense to use methods designed to cope with such complexity. System-level requirements are necessarily very abstract and ill-suited to analysis by conventional (finite state) model checkers, which require a concrete representation. However, bounded model checking for representations expressed over the theories decided by an SMT solver (so-called "infinite bounded model checking" [34]) can provide the necessary automated support [35]. Precedent for iteration between software development and system safety analysis is provided by "derived requirements"; these are requirements that emerge during software design without being directly traceable to the system requirements: the safety impacts of such derived requirements must be considered at the system level.

Another approach to safeguarding against flawed software requirements is to monitor the higher-level system safety requirements at runtime. The monitor can signal an "alarm" if these requirements are violated and other systems will ignore the outputs of a system whose monitor is signaling an alarm: the monitor transforms potential malfunction or unintended function into loss of function and prevents transitions into hazardous states. ARP 4754A explicitly discusses this type of *monitored architecture* and their reliability is examined in [21] where it is shown that the possible perfection of the monitor, unlike its reliability, is conditionally independent of the reliability of the operational system; this means that a monitor assured to some probability of perfection delivers a multiplicative improvement in system reliability. (However, we must also consider the possibility that the monitor raises the alarm unnecessarily, see [21].) Since monitors can be very simple, their assurance by DO-178B, possibly buttressed by formal methods, can plausibly deliver useful probabilities of perfection, and hence provide strong assurance for the safety of the monitored system.

2. NEW CHALLENGES

I have described the current practice in certification of aircraft software, which is based on DO-178B, and now turn to some new challenges: "new," that is, since DO-178B was introduced in 1992.

One of these has already been mentioned: the large change in methods for software development and assurance that has occurred in the last 20 years. These include model-based design (e.g., Stateflow/Simulink) object-oriented programming, formal methods, and tool-supported methods of analysis. As described earlier, DO-178B does allow "alternative methods of compliance" and even describes use of formal methods as such an alternative method. Indeed, formal methods have been used on some aircraft software [23,27,42]; the difficulty has been to gain *certification credit* for their use. Certification credit means that the alternative method meets the "intent" of some objective and can replace the traditional means of doing so; in particular, those who use formal methods would like to gain some relief from the testing requirements in DO-178B. One problem in doing so is that much of the language concerning verification in DO-178B explicitly uses the word "test." Hence, cost-effective use of formal methods really needs rather larger adjustments to DO-178B than can be accommodated within an "alternative method of compliance."

To deal with this and other emerging issues, DO-178C has been developed to augment DO-178B. This document was issued in January 2012 following more than 5 years of development (it started in March 2005).[3] It should be noted that DO-178C is essentially a series of supplements to DO-178B, amplifying and interpreting its existing guidelines, and not a replacement for it. These supplements address formal methods, object-oriented technology, model-based design and verification, and tool qualification. Daniels [9] provides insight on the development of DO-178C.

In DO-178B, tools are divided into those that could introduce an error (i.e., *development tools*, such as a faulty compiler) and those that may fail to detect an error (i.e., *verification tools*, such as an unsound static analyzer). Qualification of a development tool is very onerous, since such a tool can be used to eliminate other assurance processes (for example, compilers are usually unqualified and that is one of the reasons for requiring extensive testing of the executable code; a qualified compiler might allow this testing to be replaced by source code analysis). Verification tools are treated more lightly because they have traditionally not been used to justify elimination of other verification or development processes; DO-178C introduces an intermediate classification for verification tools that are used to justify such elimination and raises the bar on their qualification.

By similar reasoning, the formal methods supplement requires that, for certification credit, formal models must be a conservative representation of the software artifact concerned, and any analysis methods must be sound: these ensure that formal analysis may raise false alarms but will not fail to detect errors. The relationship between formal methods and testing is acknowledged to be difficult: for some purposes one or the other, but not a combination, is required for credit (recall the earlier discussion on the several "intents" served by MC/DC testing).

Model-based methods face many of the same concerns as tools and formal methods and, in addition, they tend to blur the distinctions between the various levels of requirements and between requirements and software that are central to many of the objectives of DO-178B.

Looking forward, to what might be the concerns of a possible DO-178D, I propose two topics. First, it seems that DO-178B (and presumably DO-178C also) is effective in eliminating software implementation defects, but it does so at great cost; hence, mechanization in the form of more advanced static analysis, formal methods, and automated testing are fertile topics for research, both in technology development and in the extent to which they can satisfy the intent of DO-178B objectives. Second, the available evidence points to flawed requirements as the main source of defects in software development, and to the transition between system requirements developed under ARP 4754A and software requirements under DO-178B/C as a particular vulnerability; hence improved and, in my opinion, mechanically supported methods of requirements analysis are urgently needed. Integrating these topics into a putative DO-178D in a way that supports rational analysis will be greatly assisted if the safety case implicit in DO-178B is made explicit (in particular, the argument how the objectives support the claims).

A different source of new challenges for aircraft software certification can be summarized under the heading "increased integration." Previously, separate aircraft functions were provided by fairly independent systems loosely integrated as a "federated" system. This meant that the autopilot, for example, had its own computers, replicated for redundancy, and so did the flight management system. The two system would communicate though the exchange of messages, but their relative isolation provided natural *partitioning* that limited the propagation of faults: a faulty autopilot might send bad data to the flight management system, but could not destroy its ability to calculate or communicate. Modern aircraft employ Integrated Modular Avionics (IMA) where many critical functions share the same computer system and communications network. There is naturally concern that a fault in one function could propagate to others sharing the same resources, and so partitioning must be ensured by the mechanisms (i.e., the operating system and network technology) that manage the shared resources, and this partitioning must be maintained in the presence of random faults (due to hardware failures) in the mechanisms themselves. Design of fault-tolerant IMA platforms of this kind is a very challenging exercise, as is assurance for their correctness [32]. Even given a sound IMA platform, employing it correctly to support many critical functions (or, worse, a mixture of critical and uncritical—hence unassured—functions) is another challenging exercise, as is assurance for the software that manages it (e.g., the boot-time software that specifies the configuration of an IMA platform may be hundreds of thousands of lines of XML code).

Guidelines for the development and certification of IMA are provided by DO-297 [30] but, beyond a requirement for robust partitioning, these essentially amount to the need to demonstrate that each function works correctly when running alone, and also in conjunction with the other software on a fully loaded platform. Computer scientists might wish for a more compositional (i.e., modular, component-based) approach, but this is antithetical to current certification practices. Advisory Circular 20-148 [15] does make provision for taking DO-178B qualification data for certain kinds of "reusable" software (e.g., operating systems, libraries, or protocol stacks) from one certification to another but there is no provision for the certification of any software or system in isolation. This is because experience teaches that many hazardous situations arise through unanticipated interactions, often precipitated by faults, among supposedly separate systems. Hence, the FAA certifies only complete aircraft or engines, where all potential system interactions are available for scrutiny. However, the complexity of these interactions may defy effective scrutiny; Baker [4] describes FAA concerns in this area, citing an incident in which spurious thrust reverser unlatch signals led to retraction of the leading edge slats during takeoff [41].

[3] The present paper was written before D0-178C and its supplements were published.

Thus, although it will be challenging, I think it is worth exploring and developing modular approaches to assurance and certification using the idea of "composition frameworks": these are architectural mechanisms, such as round-based synchrony, partitioning kernels, and time-triggered buses, which ensure that components can interact only through the framework. Such frameworks are already widely used in aircraft, but their benefits have not been fully formalized or exploited in certification. A properly constituted composition framework should guarantee three properties (even in the presence of faults): composability (properties of components or prior compositions are preserved when new components are added), compositionality (system properties are derived solely from component properties), and monotonicity (system properties are not reduced when a component is replaced by a superior one having more properties). Although a composition framework can ensure that component and system properties are preserved or interact only in desired ways, it is still necessary to think of all the right properties and to specify them correctly; here again, I believe that formal methods based on infinite bounded model checking can provide useful capabilities for exploring and formulating suitable properties.

Another kind of "integration" on board an aircraft is between its automated functions and the crew. The allocation of functions to systems and the presentation of these to the crew owes more to the accidents of history than to rational design; for example pitch control was automated before roll, and even today the pitch autopilot often is separate from the roll autopilot, and both are separate from the flight management system, and the autothrottle (which itself is separate from the engine controller). This leads to a complexity that presents severe human factors issues, often manifested as "automation surprises" [38] or "mode confusion" [37]. At present, these issues are not treated as part of software certification, even though they often are due to poor design choices, rather than human fallibilities. It is, however, feasible to model some these issues formally and to detect problems by model checking [6, 33].

The next level of integration lies beyond the individual aircraft. Traditionally, management of the airspace is the responsibility of air traffic control (ATC): a ground-based function. Each aircraft and its crew communicates with ATC and follows the instructions received. It is the responsibility of ATC to ensure adequate separation between aircraft and, to a first approximation, individual aircraft just fly the path assigned to them and do not manage their own separation. Onboard conflict detection and resolution systems supplement this by providing automated "resolutions" in emergencies. Current systems provide only vertical resolutions (climb or descend), but more advanced systems can change headings or speeds as well. These functions are implemented by algorithms that are distributed across the participating aircraft (with autonomous fallbacks if an "intruder" aircraft is unresponsive) [24].

By extending these onboard capabilities through a series of steps forming part of a plan known as NextGen, aircraft will become increasingly responsible for managing their own separation at both the strategic and tactical levels, employing algorithms that are distributed between multiple aircraft and the ground. Thus, the safety of one aircraft will become partly reliant on software running on other aircraft and on the ground. (This is not completely new: the crash of Korean Air flight 801 in Guam in 1997 was partly attributed to misconfiguration of a ground database [16].)

Ground ATC software has traditionally been certified to different criteria than flight software, and it seems that these must converge under NextGen. Similarly, policies and guidance for flight operations have traditionally been quite separate from aircraft and software certification, but it seems these, too, may converge under NextGen. Human factors arise again, because there may be dynamic reassignments of authority and autonomy between the aircraft crew, controllers on the ground, and automated systems distributed between the air and ground; formal modeling and analysis of these mixed-authority systems is therefore a useful undertaking [5]. To further compound the mix, there are proposals to allow unmanned military aircraft (UAVs) to use civilian airspace; these are not certified to the standards of passenger aircraft and have a poor safety record (see, for example [25, 26]).

The historical trend is that flight software doubles in size every two years [12]. Increasing integration will compound this trend as the notion of "flight software" expands from that on board individual aircraft to include the other software systems with which it interacts, whether on board other aircraft, or on the ground. If we assume that fault density (crudely, faults per some number of lines of code) is constant for any given assurance method, then we should expect the number of software-induced in-flight failure conditions of every severity to increase exponentially over time. There will still be a net increase in safety if integrated software systems are better than the systems (or lack of a system) that they displace. Nonetheless, it seems clear that aircraft software certification—and the field of software assurance in general—faces interesting challenges and is a fertile area for constructive research.

3. CONCLUSIONS

The current approach to certification of aircraft software, based on the DO-178B guidelines, seems to work rather well. But I submit that it is not known exactly how well it works, nor why. Attempts to propose alternative means of compliance and to update the existing guidelines (i.e., the supplements of DO-178C) would be assisted if an explicit argument were constructed to show how the objectives (i.e., evidence) required by DO-178B provide assurance for the system safety claims.[4] One difficulty is that system safety claims are stated probabilistically, whereas DO-178B is about correctness; I argued that this difficulty can be bridged using the idea of "possible perfection."

Increased outsourcing and other changes in the aircraft industry reduce some factors that may, implicitly, have contributed to the safety of aircraft software (e.g., organizational experience and safety culture). It therefore may be desirable that software certification should become more focused on (tool-based) examination of the actual software products (i.e., requirements, specifications, and code), and less on the processes of their development.

Although no aircraft crash has been attributed to software, there have been some incidents that should raise concern. These are invariably traced to flawed requirements,

[4]RTCA document DO-248C [31], one of the supplements to DO-178C, provides rationale for DO-178C and DO-278A objectives, but was not available at the time this paper was written.

so methods (such as infinite bounded model checking) that are capable of analyzing high-level requirements, or architectures that monitor safety properties at runtime, are worthy of consideration.

New developments in aircraft systems and air traffic management are greatly increasing the interaction among previously separate software systems, and changing the balance of autonomy and authority between the crew, ground controllers, and ground and airborne software. Certification of these distributed, mixed authority systems is a major challenge for the future.

Aircraft certification has previously considered only complete systems (i.e., aircraft), but increasing integration both inside and outside the individual aircraft surely requires a more compositional approach. A compositional approach to safety assurance is a major intellectual and practical challenge for the future.

4. ACKNOWLEDGMENTS

I am very grateful for help and information provided during the writing of this paper by Jeff Joyce of Critical Systems Labs and Ricky Butler of NASA Langley Research Center, and for ideas and suggestions by my colleague N. Shankar. I am also grateful for extensive and useful comments on the published version of the paper that I received from Dewi Daniels, Keith Hill, Peter Ladkin, and John White. I have incorporated their corrections and many of their comments in this revision to the published paper. Of course, all errors and misinterpretations are my responsibility.

5. REFERENCES

[1] B. S. Andersen and G. Romanski. Verification of safety-critical software. *Communications of the ACM*, 54(10):52–57, 2011.

[2] Australian Transport Safety Bureau. *In-Flight Upset Event, 240 km North-West of Perth, WA, Boeing Company 777-200, 9M-MRG, 1 August 2005*, Mar. 2007. Reference number Mar2007/DOTARS 50165, available at `http://www.atsb.gov.au/publications/investigation_reports/2005/AAIR/aair200503722.aspx`.

[3] Australian Transport Safety Bureau. *In-Flight Upset 154 km West of Learmonth, WA, 7 October 2008, VH-QPA, Airbus A330-303*, Dec. 2011. Aviation Occurrence Investigation AO-2008-070, Final.

[4] K. Baker. Filling the FAA guidance and policy gap for systems integration and safety assurance. In *30th AIAA/IEEE Digital Avionics Systems Conference*, Seattle, WA, Oct. 2011. The Institute of Electrical and Electronics Engineers.

[5] E. J. Bass, M. L. Bolton, K. M. Feigh, D. Griffith, E. Gunter, W. Mansky, and J. Rushby. Toward a multi-method approach to formalizing human-automation interaction and human-human communications. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 1817–1824, Anchorage, AK, Oct. 2011.

[6] E. J. Bass, K. M. Feigh, E. Gunter, and J. Rushby. Formal modeling and analysis for interactive hybrid systems. In *Fourth International Workshop on Formal Methods for Interactive Systems: FMIS 2011*, volume 45 of *Electronic Communications of the EASST*, Limerick, Ireland, June 2011.

[7] A. Bertolino and L. Strigini. Assessing the risk due to software faults: Estimates of failure rate vs. evidence of perfection. *Software Testing, Verification and Reliability*, 8(3):156–166, 1998.

[8] P. Bishop and R. Bloomfield. A methodology for safety case development. In *Safety-Critical Systems Symposium*, Birmingham, UK, Feb. 1998. Available at `http://www.adelard.com/resources/papers/pdf/sss98web.pdf`.

[9] D. Daniels. Thoughts from the DO-178C committee. In *6th IET International Conference on System Safety*, Birmingham, UK, Sept. 2011. The Institution of Engineering and Technology.

[10] S. Dekker. The criminalization of human error in aviation and healthcare: A review. *Safety Science*, 49:121–127, 2011.

[11] J. Downer. Trust and technology: The social foundations of aviation regulation. *British Journal of Sociology*, 61(1):83–106, 2010.

[12] D. L. Dvorak (ed.). NASA study on flight software complexity. Final report, NASA Office of Chief Engineer, 2009.

[13] The European Aviation Safety Agency (EASA). *Certification Specifications and Acceptable Means of Compliance for Large Aeroplanes, CS-25 and AMC-25*, July 2011. Amendment 11; available at `http://www.easa.eu.int/agency-measures/certification-specifications.php#CS-25`.

[14] Federal Aviation Administration. *RTCA, Inc. Document RTCA/DO-178B*, Jan. 11, 1993. Advisory Circular 20-115B.

[15] Federal Aviation Administration. *Reusable Software Components*, Dec. 7, 2004. Advisory Circular 20-148.

[16] W. S. Greenwell, E. A. Strunk, and J. C. Knight. Failure analysis and the safety-case lifecycle. In *IFIP Working Conference on Human Error, Safety and System Development (HESSD)*, Toulouse, France, Aug. 2004.

[17] C. Haddon-Cave. The Nimrod Review: An independent review into the broader issues surrounding the loss of the RAF Nimrod MR2 Aircraft XV230 in Afghanistan in 2006. Report, The Stationery Office, London, UK, Oct. 2009. Available at `http://www.official-documents.gov.uk/document/hc0809/hc10/1025/1025.pdf`.

[18] T. Kelly. *Arguing Safety—A Systematic Approach to Safety Case Management*. PhD thesis, Department of Computer Science, University of York, UK, 1998.

[19] P. B. Ladkin. *Computer-Related Incidents with Commercial Aircraft*. `http://www.rvs.uni-bielefeld.de/publications/compendium/`.

[20] B. Littlewood. The use of proof in diversity arguments. *IEEE Transactions on Software Engineering*, 26(10):1022–1023, Oct. 2000.

[21] B. Littlewood and J. Rushby. Reasoning about the reliability of fault-tolerant systems in which one component is “possibly perfect”. *IEEE Transactions on Software Engineering*, 2011. Accepted for publication.

[22] E. Lloyd and W. Tye. *Systematic Safety: Safety Assessment of Aircraft Systems*. Civil Aviation Authority, London, England, 1982. Reprinted 1992.

[23] S. P. Miller, M. W. Whalen, and D. D. Cofer. Software model checking takes off. *Communications of the ACM*, 53(2):58–64, Feb. 2010.

[24] C. Muñoz, R. Butler, A. Narkawicz, J. Maddalon, and G. Hagen. A criteria standard for conflict resolution: A vision for guaranteeing the safety of self-separation in NextGen. Technical Memorandum NASA/TM-2010-216862, NASA, Langley Research Center, Hampton VA 23681-2199, USA, October 2010.

[25] National Transportation Safety Board, Washington, DC. *Safety Recommendation A-07-70 through -86*, Oct. 2007. Available at `http://www.ntsb.gov/Recs/letters/2007/A07_70_86.pdf`.

[26] National Transportation Safety Board, Washington, DC. *Safety Recommendations A-07-65 through -69*, Oct. 2007. Available at `http://www.ntsb.gov/recs/letters/2007/A07_65_69.pdf`.

[27] I. M. O'Neil, D. L. Clutterbuck, P. F. Farrow, P. G. Summers, and W. C. Dolman. The formal verification of safety-critical assembly code. In W. D. Ehrenberger, editor, *Safety of Computer Control Systems (SAFECOMP '88)*, pages 115–120, Fulda, Germany, 1988. International Federation of Automatic Control, IFAC Proceedings Series No. 16.

[28] Requirements and Technical Concepts for Aviation (RTCA), Washington, DC. *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*, Dec. 1992. This document is known as EUROCAE ED-12B in Europe.

[29] Requirements and Technical Concepts for Aviation (RTCA), Washington, DC. *DO-254: Design Assurance Guidelines for Airborne Electronic Hardware*, Apr. 2000.

[30] Requirements and Technical Concepts for Aviation (RTCA), Washington, DC. *DO-297: Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations*, Nov. 2005. Also issued as EUROCAE ED-124 (2007).

[31] Requirements and Technical Concepts for Aviation (RTCA), Washington, DC. *DO-248C: Supporting Information for DO-178C and DO-278A*, Dec. 2011.

[32] J. Rushby. Bus architectures for safety-critical embedded systems. In T. Henzinger and C. Kirsch, editors, *EMSOFT 2001: Proceedings of the First Workshop on Embedded Software*, volume 2211 of *Lecture Notes in Computer Science*, pages 306–323, Lake Tahoe, CA, Oct. 2001. Springer-Verlag.

[33] J. Rushby. Using model checking to help discover mode confusions and other automation surprises. *Reliability Engineering and System Safety*, 75(2):167–177, Feb. 2002. Available at `http://www.csl.sri.com/users/rushby/abstracts/ress02`.

[34] J. Rushby. Automated formal methods enter the mainstream. *Communications of the Computer Society of India*, 31(2):28–32, May 2007. Special Theme Issue on Formal Methods edited by Richard Banach. Archived in Journal of Universal Computer Science Vol. 13, No. 5, pp. 650–660, available at `http://www.jucs.org/jucs_13_5`.

[35] J. Rushby. A safety-case approach for certifying adaptive systems. In *AIAA Infotech@Aerospace Conference*, Seattle, WA, Apr. 2009. American Institute of Aeronautics and Astronautics. AIAA paper 2009-1992; available at `http://www.csl.sri.com/users/rushby/abstracts/aiaa09`.

[36] J. Rushby. Software verification and system assurance. In D. V. Hung and P. Krishnan, editors, *Seventh International Conference on Software Engineering and Formal Methods (SEFM)*, pages 3–10, Hanoi, Vietnam, Nov. 2009. IEEE Computer Society.

[37] N. B. Sarter and D. D. Woods. How in the world did we ever get into that mode? Mode error and awareness in supervisory control. *Human Factors*, 37(1):5–19, 1995.

[38] N. B. Sarter, D. D. Woods, and C. E. Billings. Automation surprises. In G. Salvendy, editor, *Handbook of Human Factors and Ergonomics*. John Wiley and Sons, second edition, 1997.

[39] Society of Automotive Engineers. *Aerospace Recommended Practice (ARP) 4754: Certification Considerations for Highly-Integrated or Complex Aircraft Systems*, Nov. 1996. Also issued as EUROCAE ED-79; revised as ARP 4754A, December 2010.

[40] Society of Automotive Engineers. *Aerospace Recommended Practice (ARP) 4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, Dec. 1996.

[41] South African Civil Aviation Authority, Incident Investigation Division. *Boeing B747-400 G-BYGA Group 'A' Leading Edge Slats Retracted on Takeoff from O. R. Tambo Airport, South Africa, 11 May 2009*, June 2010. Serious Incident Investigation Report CA18/3/2/0717.

[42] J. Souyris, V. Wiels, D. Delmas, and H. Delseny. Formal verification of avionics software products. In *Sixteenth International Symposium of Formal Methods, FM 2009*, volume 5850 of *Lecture Notes in Computer Science*, pages 532–546, Eindhoven, The Netherlands, Nov. 2009. Springer-Verlag.

[43] UK Air Investigations Branch. *Report on the incident to Airbus A340-642, registration G-VATL en-route from Hong Kong to London Heathrow on 8 February 2005*, 2007. Available at `http://www.aaib.gov.uk/publications/formal_reports/4_2007_g_vatl.cfm`.

[44] A. Wassyng, T. Maibaum, M. Lawford, and H. Bherer. Software certification: Is there a case against safety cases? In R. Calinescu and E. Jackson, editors, *16th Monterey Workshop: Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, volume 6662 of *Lecture Notes in Computer Science*, pages 206–227, Redmond, WA, 2011. Springer-Verlag.