Perspectives on Software Safety Case Development for Unmanned Aircraft

Ewen Denney and Ganesh Pai
SGT / NASA Ames Research Center
Moffett Field, CA 94035, USA
Email: {ewen.denney, ganesh.pai}@nasa.gov

Ibrahim Habli
Department of Computer Science
University of York, York, UK
Email: Ibrahim.Habli@cs.york.ac.uk

***Abstract*—We describe our experience with the ongoing development of a safety case for an unmanned aircraft system (UAS), emphasizing autopilot software safety assurance. Our approach combines formal and non-formal reasoning, yielding a semi-automatically assembled safety case, in which part of the argument for autopilot software safety is automatically generated from formal methods. This paper provides a discussion of our experiences pertaining to (a) the methodology for creating and structuring safety arguments containing heterogeneous reasoning and information (b) the comprehensibility of, and the confidence in, the arguments created, and (c) the implications of development and safety assurance processes. The considerations for assuring aviation software safety, when using an approach such as the one in this paper, are also discussed in the context of the relevant standards and existing (process-based) certification guidelines.**

***Keywords*-Software safety; Safety cases; Unmanned aircraft; Formal methods; Aviation software.**

I. Introduction

The approval process for deploying airborne software largely occurs via *certification*, a core activity in developing safety-critical systems. In aviation, this occurs through demonstrating compliance with aerospace standards and regulations (e.g., DO-178B [1]) to a government appointed authority, e.g., the U.S. Federal Aviation Administration (FAA) or the European Aviation Safety Agency (EASA).

Via *prescriptive certification*, especially in civil aviation, developers demonstrate that a software system is acceptably safe by appealing to the satisfaction of a set of process objectives that the safety standards require for compliance. The means for satisfying these objectives are often tightly defined within the standards. In general, such prescriptive safety standards offer useful guidance on "good practice" software engineering methods, safety analysis techniques, and the way in which factors such as independence in the process can improve confidence. However, a fundamental limitation lies in the observation that good tools, techniques, and methods do not necessarily lead to the achievement of a specific level of integrity/assurance, often expressed in terms of failure rates. A correlation between the prescribed techniques and the failure rate of systems has not been demonstrated [2].

Increasingly, consideration is being given to *goal-based certification*, where the corresponding standards require the submission of a safety argument that communicates how evidence, e.g., generated from testing, analysis and review, satisfies claims concerning safety. This is commonly referred to as a *safety case* [3]; the counterpart for software is the *software safety case* [4]. The requirement for a safety case has been central in goal-based standards, particularly in domains such as defense, rail, and oil & gas. The revision of DO-178B, i.e., DO-178C, includes a new note concerning *assurance arguments*[1] as an alternative method to show that system safety objectives are satisfied. Similarly, the interim guidance for approving unmanned aircraft system (UAS) operations includes the use of "alternate methods of compliance", specifically referring to a safety case with sufficient data, as a means for possible approval [6, Section 5.0].

Although there is some general guidance on the use of safety cases in aviation [7], and for airborne software [8], to our knowledge, there is little documented experience on their application to UAS [9], given the existing (and upcoming) framework for approval. We anticipate that reporting the perspectives from developing a system/software safety case for a UAS might, eventually, aid the formulation of such guidance; however, that is not the main aim of this paper.

Rather, it is to examine (i) whether a safety case can transparently and coherently communicate assurance by reconciling heterogeneous safety-relevant information and diverse reasoning, and (ii) the extent to which software safety assurance can be improved by the development of an explicit software safety case, particularly from a systems perspective. In this paper, we briefly describe our methodology for developing a safety case for an experimental UAS with a focus on the safety assurance of the autopilot software. We then report the insights gained / lessons learned from the perspectives of safety case design, comprehensibility of the arguments, quantitative confidence assessment, and the implications from existing processes and standards.

II. The Swift UAS

We are interested particularly in assuring the safety of the airborne system in the *Swift* unmanned aircraft system (UAS) being developed at NASA Ames. The UAS comprises

[1]Conceptually, a safety case can be considered as a specialization of an assurance case [5].

978-1-4673-1625-5/12/$31.00 ©2012 IEEE

the electric Swift Unmanned Aerial Vehicle (UAV), dual redundant ground control stations (GCS) and communication links. The UAV can fly autonomously, in different modes, e.g., computer in control (CIC), following an uploaded or a pre-programmed *nominal flight plan*; it can be also controlled by a pilot on the ground. An *off-nominal flight plan* describes a protocol for managing contingencies, e.g., failures in the primary pilot system, to ensure that any crash that may occur is "on range".

The flight software onboard the UAV is implemented as a collection of loosely-coupled modules (e.g., the autopilot, communication interfaces, configuration scripts, etc.) that are defined on top of, and interface with, the *Reflection* system: a multi-component, event-driven, real-time, configurable software framework. In turn, this system runs on top of a *physics library* which itself runs on an embedded real-time operating system. The onboard autopilot is involved, in part, in the contingency management system (CMS); therefore, assuring its (functional) safety is especially relevant. The autopilot software has a modular design and consists of the flight management system (`FMS`) and the controller (`AP`) modules. Both of these are involved in controlling the flight surfaces, and therefore, aircraft movement.

It is important to note that the Swift UAS design, which is far along in its development lifecycle, has incorporated safety considerations and reuses specific functionality from a predecessor vehicle. However, certain functionality, e.g., the autopilot, is yet to be fully designed. This presents us with the opportunity to influence its design via the application of our safety assurance methodology, described subsequently.

III. Safety Assurance

A. Methodology

Our approach (Figure 1) adopts goal-based argumentation for linking evidence, e.g., results of software verification, to claims that hazards, identified from safety analysis, are mitigated. The system safety and safety argumentation processes are intertwined with system development, start at the level of the UAS, and are repeated at the software level.

Through our approach, we (i) explicitly consider diverse evidence, reasoning, assumptions and context in the safety analysis and argument (ii) automatically generate parts of the software safety case from formal verification, i.e., proofs of correctness, thereby both integrating formal reasoning into safety argumentation and providing a significantly greater level of detail in the argument for software (iii) automatically assemble the argument for software safety into that of safety of the system, and (iv) explicitly consider the confidence that can be placed in the safety case being put forth, through uncertainty assessment.

We use the Goal Structuring Notation (GSN) [3] to document the safety case. In brief, GSN is a graphical notation for representing arguments in terms of basic elements such as *goals* (claims), *context*, *evidence*, *assumptions*

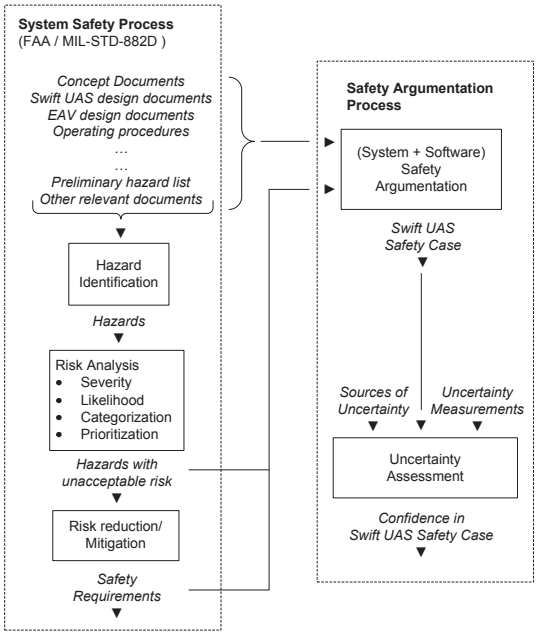

Figure 1. Safety assurance methodology: key activities and data flow.

and *justifications*. Arguments are created in GSN by linking these elements using two main relationships: *supported by*, and *in context of*, to form a goal structure. We also use the AutoCert tool [10] to support the formal verification of software through automatic theorem proving. It produces a proof, along with supporting logical axioms and function specifications that have been used. This proof can be automatically transformed into a safety case fragment, using a tool that we developed in-house at NASA Ames. This tool also automatically assembles the software safety case into the system-level safety case. We describe how we applied our approach to the Swift UAS subsequently in this section.

B. Safety Analysis

We base the system safety process on the framework of a safety risk management plan [11], so as to include safety considerations into system/software development.

The process begins with hazard identification and risk analysis, i.e., as preliminary hazard analysis (PHA) at the system level; then it is repeated as subsystem hazard analysis (SSHA) at the software level. We applied a variety of hazard identification and analysis techniques such as failure modes and effects analysis (FMEA) and fault tree analysis (FTA). To refine the hazard analysis, and to manage the wider context of safety, i.e., safety implications arising from interactions, and deviations in processes and procedures, we also explicitly characterized the heterogeneous sources of evidence, assumptions and context that could be used to support a claim of safety, e.g., concepts of operations, operating procedures, assumptions made in theoretical models

(of flight control / aerodynamic stability), simulations and computational models, etc.

Then, we identified unacceptable risks by categorizing hazards based on a combination of the severity of consequences, and the likelihood of occurrence. Subsequently, we defined mitigation measures/controls, and we specified the corresponding (system and software) safety requirements.

For instance, *Unintended pitch down during cruise* is a system-level hazard to which the autopilot software can contribute by failing to correctly compute the relevant parameters, e.g., angle of attack, and/or the output values for the flight surface actuators. These are software failure modes, and we specify the corresponding (software) safety requirements to negate these failure modes e.g., "The autopilot shall accurately compute the correct angle of attack", and "The autopilot shall correctly compute actuator outputs".

C. Arguing Safety of the System

The outcome of the system safety process, in effect, triggers the safety argumentation process (Figure 1).

Abstractly, the main steps in the safety argumentation process are to (a) define safety claims, e.g., that a specific hazard is eliminated, and (b) identify, select, and link the evidence, which support the claims made, via a structured argument that can, on analysis, be agreed upon as acceptable and trustworthy. The top-level claim (goal) in our safety case is: *The Swift UAS is safe in the context of the defined mission, in the specified configuration, on the defined range where it is to be operated, and under the defined weather conditions.*

To develop this claim our strategy is, primarily, to argue that all identified hazards across all operating phases have been mitigated. In turn, we develop the claims of hazard mitigation by argument over the UAV subsystems (e.g., the avionics subsystem, of which the autopilot software is a part).

D. Arguing Software Safety in the System Context

The autopilot software safety case (ASSC) contains a justification for the claim that the autopilot is correct (functional safety), and it makes explicit the heterogeneity of context, assumptions and evidence inherent in a safety claim, e.g., Figure 2(a) shows a small excerpt from the ASSC where we argue the safe computation of the angle of attack parameter by justifying the claim that the autopilot computes it accurately and correctly.

This justification is through an argument structure which links diverse evidence, i.e., a proof of correct implementation, results of reviewing the corresponding specification, data sheets for the air-data (pitot) probe, and the results of wind tunnel experiments to calibrate the probe. Note that the argument leg that supports the claim of correct implementation (G2.2), as we have shown it in Figure 2(a), abstracts a significantly more detailed argument fragment.

The items of evidence in the ASSC are, in part, generated from formally verifying the autopilot software implementation against a mathematical specification, using AUTOCERT. The specification contains assumptions, e.g., about the aircraft state and flight plan, and it formalizes the software requirements, a subset of which are derived from the system safety requirements. Formal verification takes place in the context of a logical domain theory, i.e., a set of axioms and function specifications. Axioms can be either assumed to be correct, or they can be inspected, or they can be tested against a computational model which, itself, is inspected.

Through this approach, we are able to integrate formal reasoning into the construction of a (software) safety case. Conversely, formal reasoning makes use of safety-relevant information which has not itself been derived using formal methods. This can take the form of simplifying assumptions which are experimentally justified, or appeals to expert judgment, e.g., that a parameter is within safe bounds, that an error is within an acceptable range, or that one subsystem is similar to another (and therefore has equivalent safety-related properties).

We automatically transform the output of AUTOCERT into a safety case fragment, and then merge it into the upper-level (system) safety case by replacing the relevant, overlapping, goals in the latter with the auto-generated argument fragment (containing the proof as evidence). If safety case fragments have been created prior to the construction of a proof (as might be reasonably expected), they also can be converted into formal specifications for input to AUTOCERT.

E. Assurance of the Arguments

Assurance can be defined as justified confidence in a property of interest [12]. Despite the explicit consideration of diverse evidence and formal reasoning in the software safety case, subjectivity is inherent in the structure of the argument and its supporting evidence. To assure that sufficient confidence can be placed in the arguments made, our approach is based on quantitative uncertainty assessment [13], which forms part of the broader safety process (Figure 1). Confidence assessment provides feedback for improving the safety case during its evolution (although it is not explicitly highlighted in Figure 1).

We begin by identifying the sources of uncertainty in the argument, following which we quantify the uncertainty introduced by these sources. We use data where available (in the case of aleatory uncertainties), and subjective judgment for epistemic uncertainties. Thereafter, we aggregate the quantified values into an assessment of argument/claim confidence via probabilistic modeling. More specifically, we model the identified sources of uncertainty as discrete random variables (r.v.), and we characterize the confidence in the overall argument as the joint distribution of the r.v., using Bayesian Networks (BN).

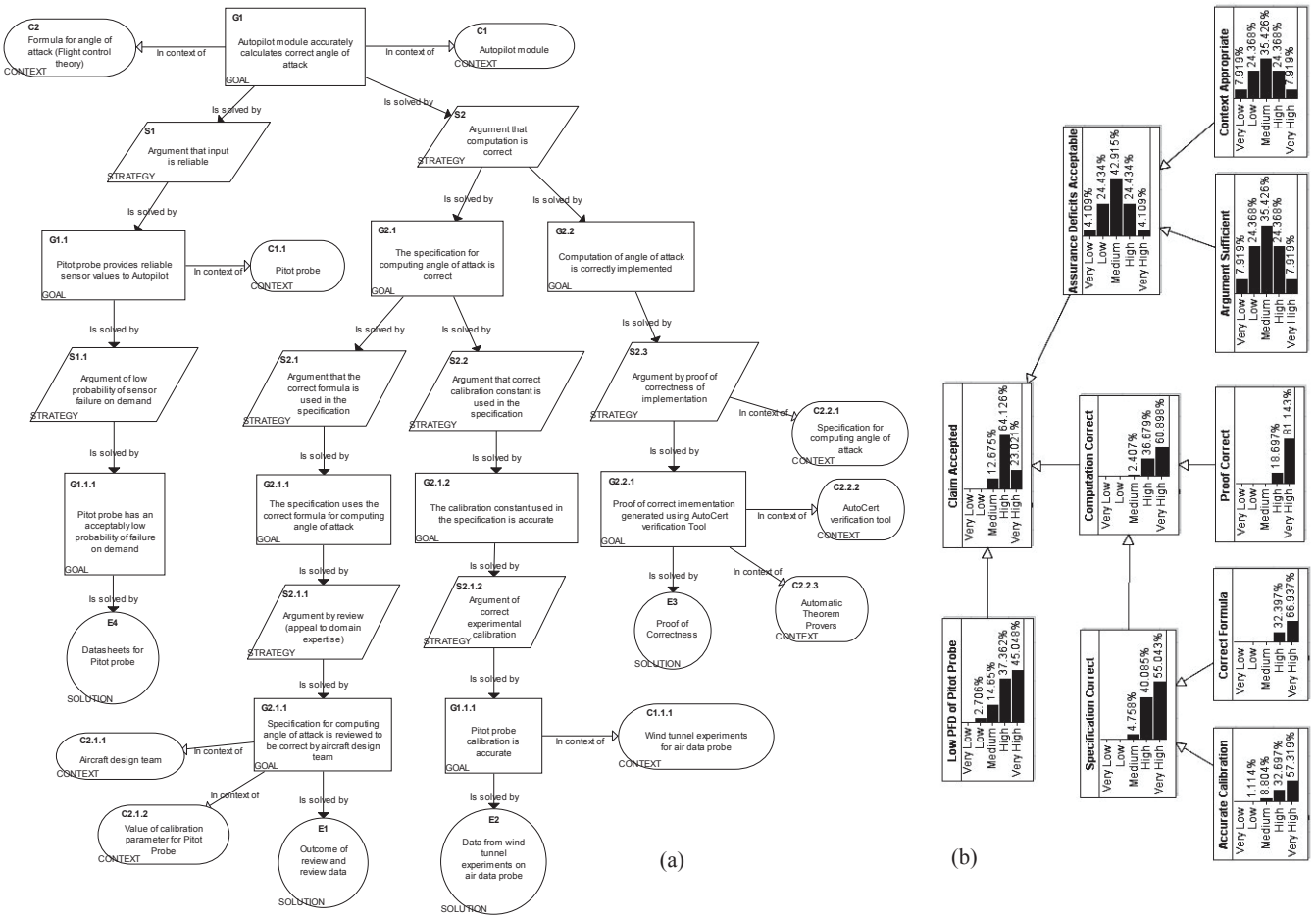

Figure 2. (a) Safety case fragment showing an (abridged) argument for safe computation of angle of attack (b) Confidence quantification.

To illustrate, the BN in Figure 2(b) gives the prior confidence in the argument of Figure 2(a). The BN structure mirrors the initial decomposition of the top-level claim and it also captures the uncertainty in the argument formulation, i.e., the *assurance deficits* [9]. Each node in the BN and its associated distribution characterizes the different sources of uncertainty in the argument. The BN completely specifies the prior confidence in the argument by modeling the uncertainty that the claim is accepted, conditional on the uncertainties identified for the argument.

Thus, stochastic (aleatory) uncertainty in the sensor values is influenced by the failure probability (rate) of the pitot probe. This is quantifiable, e.g., through statistical testing of the sensor. On the other hand, uncertainty in the correctness of the specification is both aleatory and epistemic in nature. The former appears since a parameter in the function specification is obtained from wind tunnel calibration of the pitot probe and due to the uncertainty in the calibration experiments. We take the confidence level of the calibration experiments as the quantitative confidence value for the corresponding BN node, Accurate Calibration. Epistemic uncertainty in the specification correctness is given by subjectively quantifying the confidence that the correct formula has been used [13], after domain experts review the specification against flight control theory.

Even though a formal proof of correct implementation is available, there is uncertainty along the verification chain induced through a combination of uncertainties surrounding the assumptions made, e.g., in the specifications, and the tools being used. We gauged the uncertainty that the proof is correct, using subjective judgment based on feedback from the tool developers. Thus, the node Proof Correct in the BN of Figure 2(b) conveys the notion that there is, a priori, *very high* confidence that the proof should be trusted.

There also exists uncertainty in the sufficiency of the sub-claims, i.e., whether the sub-claims G1.1, G2.1, and G2.2 are appropriate and sufficient to infer the parent claim G1. Indeed, arguments can only ever be deemed "complete" (or compelling) insofar as the uncertainty in the reasoning is sufficiently low. For instance, the node Argument Sufficient indicates that comparatively lower confidence exists in the sufficiency of the argument, adjudged as *medium*. Uncer-

tainty also exists whether the claims made and the strategies used to decompose the claims have been applied in the appropriate context. Again, these uncertainties are epistemic and we quantified it subjectively.

Based on the specified distributions and the conditional independence assumptions as encoded by the BN structure, the results of probabilistic modeling would suggest that *high confidence* be placed in the overall argument (shown by the distribution on the node Claim Accepted). However, this evaluation is only an initial step towards confidence assessment and several challenges exist (discussed subsequently) associated with the quantification, interpretation and validation of the results from the model.

IV. Discussion

Thus far, we have described our experience with software safety case development in terms of our methodology for safety analysis and argumentation as applied to the autopilot software in the Swift UAS.

In particular, the first two authors performed the safety analysis in close cooperation with the Swift UAS engineering team. Subsequently, they also constructed the manually created part of the safety case. In doing so, domain knowledge that was previously implicit was made explicit and classified as *assumption*, *justification* or *evidence*, as appropriate, for use in the safety case. The functional safety requirements were formalized, in part, by the first author who also defined the axioms and function specifications required for formal verification. The third author functioned as the external independent safety case expert for evaluating (and subsequently enhancing) the arguments made. Then, we presented the safety case to the Swift UAS engineering team (the designers and the range safety officer) for their feedback.

In this section, we reflect upon the insights gained and lessons learned from several perspectives.

A. Safety Argument Comprehension

Our initial assumptions in the choice of a graphical notation, such as the GSN, for documenting the ASSC and the Swift UAS safety case were that a graphical notation would be intuitively easier to comprehend, and enable us to better communicate the reasoning used in the arguments therein. However, our first experience through this exercise suggests that a tabular equivalent representation is also useful to have and, in some situations, might be better received.

We hypothesize that this is due, in part, to the (relatively small) learning curve associated with a new notation, the unfamiliarity of the Swift UAS engineering team with the syntax and semantics of the GSN notation, and the prevalence of tabular representations for capturing development artifacts such as requirements, hazard logs, hazard analysis results, traceability, etc. Constructing a tabular equivalent for an argument represented using the GSN is straightforward, but presents the potential for some misinterpretation if table semantics are not well specified.

We also observed that safety case fragments containing hazard mitigation arguments over an architectural breakdown were, sometimes, misinterpreted as being equivalent to other graphical notations in safety analysis that model failure conditions over architectures, e.g., fault trees and event trees.

An additional impediment that we noted, to better comprehending the arguments as we presented them, was the need to query the arguments and answer "what-if" questions. Although this is desirable in terms of guiding design for dependability/safety, identifying shortcomings/improvements in the system and, practically, also for facilitating navigation through large goal structures, the purpose of the safety argument is primarily to convey that which has been achieved to assure safety. We hypothesize that this arose, again in part, from mistaking the semantics of the GSN notation with that of notations which permit such analyses, e.g., fault trees, and unfamiliarity, in general, with the safety case concept.

The engineers also preferred to view the safety case as an information log providing a transparent record that safety concerns have received sufficient consideration, rather than as an argument assuring safety. Indeed, one point made during the feedback we received, was that flight tests are the ultimate evidence of system safety. This is valid, but can be reconciled with our methodology by observing the following: since the Swift UAS is undergoing development, our safety case is *interim* in the safety case lifecycle [14]. By itself, it is insufficient to *completely* justify the top-level claim (Section III-C), nor should that be expected of it. Rather, the evidence of safe flight and other evidence from operation (which is contingent on mission specific configurations and weather conditions) forms part of the *operational safety case* [14], which we have yet to create.

A key lesson learned here is that communicating the safety argument in a comprehensible way to the relevant stakeholders requires a mechanism that (a) well abstracts the data that are not relevant to the stakeholders, e.g., through the choice of a modular argument, and (b) mainly presents the information relevant for their role in system development and operation.

B. Safety Argument Design

Our assurance approach was mainly to address all the identified hazards; therefore, we adopted a hazard-directed style of argumentation, an instantiation of the *hazard-directed breakdown pattern* [15]. Since we are also concerned with the safety of software, our safety case includes explicit *correctness arguments* to demonstrate and justify that software contributions to the identified system hazards are acceptable. Here, the evidence comprises formal proofs that the code correctly implements the formalized software safety requirements, and where the latter are functional safety requirements that negate the hazards.

However, like the design of software architectures, an argument structure varies depending on the attributes that it needs to satisfy. Whereas architectures are designed to satisfy quality attributes such as performance, reliability and modifiability, arguments are designed to satisfy various attributes such as compliance, comprehensibility, validity, and maintainability. Reflecting on the experience gained, the safety case we developed for the Swift UAS is not explicitly modularized, in comparison with those created using the modular, contract-based safety case concept [8].

Thus, our safety case structure would have been different (more modular) if, in addition, we were concerned with compliance, maintainability, and reducing the cost of re-certification (in the event of change).

An important objective for a modular safety case is to establish a correspondence between the modules in the safety case and the components in the system and software architecture. For our safety case, and since our concern was mainly to show how software contributions to hazards were acceptable, splitting the structure of the safety case into slices of arguments, each covering one hazard, offered a better strategy for addressing that concern (i.e., in terms of highlighting traceability between hazards and software behavior over correspondence between argument structures and software components).

Similarly, the overall structure of the safety case for the Swift UAS would have been different if mere compliance was our objective, e.g., compliance with many safety standards in the UK requires demonstrating that all residual risks are *As Low As Reasonably Practicable* (ALARP)[2]. This would require a cost-benefit analysis, specifically to show that deploying additional risk reduction measures would require grossly disproportionate costs. We did not explicitly consider the costs of risk reduction in our safety case since the ALARP principle is not currently applicable to the Swift UAS. However, it is worth noting that cost-benefit analysis is part of the design trade-off that the development team performed during systems engineering.

C. Assessment of Confidence

1) Challenges: We selected a Bayesian paradigm as the basis for quantitative confidence analysis since it affords a common probabilistic framework in which to reason about both subjective information and quantitative data. However, several issues exist relevant to quantification, validation and interpretation of the BN model.

First, there is a need to justify the basic BN structure and the assumptions of conditional independence. This could be achieved, in part, by automatically generating the BN from the GSN-based safety argument, where for each source of uncertainty identified, a corresponding node (or nodes) exists in the BN. Next, specifying the prior probabilities for the leaf nodes (nodes with no outgoing arcs) is straightforward where empirical data is available (e.g., the node Accurate Calibration, in Figure 2(b)).

When only subjective judgment is available (e.g., the node Correct Formula in Figure 2(b)), quantifying confidence and selecting an appropriate prior distribution is problematic despite extensive research on belief elicitation methods [16]. One way to address this issue, we believe, is to identify relevant metrics using techniques such as the Goal-Question-Metric (GQM) method [17], and to correlate these metrics to confidence levels based on a defined quality model, e.g., we hypothesize that a metric such as coverage (by a safety argument) of hazards (in a hazard list) would correlate with confidence in the sufficiency of the argument.

For intermediate nodes (i.e., nodes with both incoming and outgoing arcs, e.g., the node Computation Correct in Figure 2(b)), we use a parametric form to specify the conditional prior probability distribution (not shown), where some of the parameters model the *strength of correlation* between the intermediate nodes and its immediate parents. Here, greater investigation is required to justify the parameters used, since these are also specified subjectively. Assuming that the strategies used to decompose goals are viewed as being equally important, using equal weights appears to be a reasonable way forward.

Finally, the model must be extended for interpreting its results reasonably as a basis for decision making, e.g., by defining the thresholds at which the confidence assessed is considered sufficient as to accept or reject the claims made.

2) Towards Internal Measures of Quality: The above items present a promising avenue by which to define an integrated internal measure for quality, e.g., by extending the measure of hazard coverage, say, to account for argument validity expressed as a confidence level. As a preliminary step along these lines, we defined an initial set of measures (including coverage), for objectively evaluating our approach. Table I gives the results of applying these measures to the Swift UAS safety case. The measures indicate reasonably high, but not complete, coverage of the hazards and corresponding high-level requirements considered, i.e., about 73% and 87% respectively. This reflects the fact that not all claims corresponding to the hazards and high-level requirements, in the argument fragment, have been *completely developed*, i.e., end in available evidence.

On the other hand, we measure perfect coverage of low-

Table I
QUANTITATIVE MEASURES FOR THE SWIFT UAS SAFETY CASE

Measure	Value
Total number of claims	144
Coverage of considered hazards	0.7344
Coverage of high-level safety requirements	0.8667
Coverage of low-level safety requirements	1
Code covered by auto-generated claims	0.9215

[2]http://www.hse.gov.uk/risk/theory/alarp.htm

level requirements by claims, since all corresponding claims are both auto-generated and terminate in evidence. However, the extent to which the low-level requirements are actually covered (by the verification) is not perfect. This is reflected in the last row of Table I, where about 92% of the code is covered by auto-generated claims. This measure indicates that there are parts of the code which were not reasoned about, were not covered by a requirement, and therefore remained unproven. To use these initial measures for quality assessment, we require a quality model, similar to that used to interpret the outcome of uncertainty assessment.

D. Implications on Processes

The existing framework used by the Swift UAS engineering team for systems development is fairly mature, based on extensive engineering knowledge and experience with developing previous UAVs, and the development of certain functionality is relatively rapid. Thus, there is a need to tailor the safety methodology such that well-defined interfaces exist to the system development process, so that the safety process can keep up with development and actively influence it. To this end, having pre-defined intervals in the development process to synchronize with the safety process, appears to be a reasonable way forward. Another alternative, is to synthesize the development process from the safety activities, as in [18].

The process as we applied it and the resulting arguments are "heavyweight" in proportion to the development costs and effort involved. Nevertheless, its application is justified largely because loss of the Swift UAV is an intolerable risk. This is due, in part, to the uniqueness of its avionics system, the modifications that were made to the airframe for mission-specific purposes, and the extent of research investments in the UAV. However, for smaller UAV, the loss of aircraft may not always be an intolerable risk. Thus, there is a need for a *lightweight* version of our approach, wherein the effort involved is appropriately proportional to the costs.

The development team found software hazard analysis to be useful in understanding and appreciating the subtleties of software contributions to system hazards. The engineers also adopted and integrated the procedures for system/software hazard analysis into their development process such that safety is proactively included as a consideration in the development of subsequent functionality, e.g., the electric-battery management system.

E. Implications of Existing Guidelines/Standards

Many items of evidence used in the ASSC (Section III) can be mapped to lifecycle data required for compliance with DO-178B, e.g., evidence E1 of requirements review in the argument (Figure 2(a)), is mapped to Objective 2 in Table A4 in DO-178B (i.e., "Low level requirements are accurate and consistent"). The added value of our software safety argument is that it explains how the evidence, potentially compliant with DO-178B, can provide sufficient assurance, particularly with regard to claims concerning how software behavior relates to specific system hazards (e.g., the link between incorrect autopilot behavior at the software level and a loss event at the system level).

The areas of a hazard directed safety argument which would be least supported by DO-178B evidence are those related to the analysis of hazardous software failure modes. This relates to the relationship between assuring safety and demonstrating correctness for software. Specifically, the DO-178B guidance implies that safety analysis is a system-level process. As such, the role of software development is to demonstrate correctness against requirements, including safety requirements allocated to software, as generated from the system-level processes. The process of refining and implementing these requirements at subsequent stages does not involve any explicit application of software hazard analysis. To this end, in DO-178B, the link between software behavior and the required safety properties of the system is implicit.

However, others [9] have argued that hazard analysis should be applied at the software level for each development stage (e.g., performing hazard analysis at the software architecture, design and source code stages). The reasoning underlying their approach is that errors could be introduced at each software development stage, which have the potential to lead to hazardous failure conditions at the system level. As such, it is important to provide assurance, through explicit software safety arguments, that these software errors have been identified and managed.

In the software assurance for the Swift UAS, we performed hazard analysis down to the software architecture level, identifying how software components and interactions can contribute to system hazards. Then we defined safety requirements, and allocated them to the components and interactions. Subsequently, we demonstrated safety assurance by appealing to the correctness of the implementation of these interactions and components against the allocated, and formalized, safety requirements (e.g., correctness of the autopilot software against allocated safety requirements).

The criterion for deciding that no further hazard analysis was needed at subsequent stages, was that a software component was simple enough (as gauged by the developers) that requirements could be formulated to demonstrate desired behavior and the absence of unintended behavior.

Finally, the explicit representation of a software safety argument is useful for developers wishing to use alternative means for compliance, e.g., by generating evidence from formal mathematical analysis rather than from testing. Where alternative approaches are used, a reviewer must be convinced of the relevance and suitability of the alternative evidence. Previous experience [19] has shown that this is much easier to achieve using a documented safety argument. This issue is particularly important now, as DO-178B has been updated (to DO-178C), with guidance on formal

methods, and to allow assurance arguments as an alternative method for establishing compliance. Although compliance with DO-178B is currently not required for the Swift UAS software, this paper provides an example and assessment of how assurance arguments can be developed for airborne software, justifying how evidence generated from formal methods can be used to support claims about safe software behavior.

V. Conclusions and Outlook

Our perspectives in this paper are mainly based on an interim safety case, which represents only a small part of the overall Swift UAS safety case (which, in turn, includes arguments assuring the safety of the ground system, the communication infrastructure, and UAS operation, besides those for the airborne system and its software). We note that the interim safety case alone is insufficient to make any categorical claims about safety improvement of the system. Rather, the safety argument is intended to communicate that certain specific claims made can be defended with reasonable confidence based on the evidence, the assumptions and the context supplied therein. Although we created the safety case, in part, by using a formal verification tool, the methodology applies more generally to include other safety and dependability techniques.

Based on the feedback received, we are actively pursuing research directions that are expected to improve, in general, the practical application of safety cases, and the Swift UAS safety case in particular. This includes improving the modularity of the argument via abstraction to aid argument comprehension, assessing the soundness of the manually created argument fragments by systematically and more extensively evaluating the argument for fallacies [20], better characterizing the diversity of available evidence to facilitate automatic assembly of the safety argument, and tailoring the methodology to make it *lightweight*.

The need to manage and reconcile diverse information in both the system and software safety cases becomes apparent from the perspective of not only safety, but also compliance to airworthiness requirements for operating a UAS [6], where the overarching goal is to show that a level of safety equivalent to that of manned operations exists [21]. We believe that the approach and experience documented in this paper is a step towards that end.

Acknowledgments

NASA contract NNA10DE83C funded this work. We also thank Mark Sumich and Corey Ippolito for their feedback.

References

[1] RTCA SC-167 and EUROCAE WG-12, "Software Considerations in Airborne Systems and Equipment Certification," DO-178B/ED-12B, 1992.

[2] F. Redmill, "Safety integrity levels – theory and problems, lessons in system safety," in *Proc. 18th Safety-Critical Sys. Symp.*, Feb. 2010.

[3] Goal Structuring Notation Working Group, "GSN Community Standard Version 1," Nov. 2011. [Online]. Available: http://www.goalstructuringnotation.info/

[4] R. Weaver, "The safety of software – constructing and assuring arguments," Ph.D. dissertation, Dept. of Comp. Sci., Univ. of York, 2003.

[5] R. Bloomfield and P. Bishop, "Safety and assurance cases: Past, present and possible future – an Adelard perspective," in *Proc. 18th Safety-Critical Sys. Symp.*, Feb. 2010.

[6] K. D. Davis, "Unmanned Aircraft Systems Operations in the U.S. National Airspace System," Interim Operational Approval Guidance 08-01, FAA Unmanned Aircraft Systems Program Office, Mar. 2008.

[7] European Organisation for the Safety of Air Navigation, *Safety Case Development Manual*, 2nd ed., EUROCONTROL, Oct. 2006.

[8] J. Fenn, R. Hawkins, P. Williams, and T. Kelly, "Safety case composition using contracts - refinements based on feedback from an industrial case study," in *Proc. 15th Safety-Critical Sys. Symp.*, Feb. 2007.

[9] R. Hawkins, K. Clegg, R. Alexander, and T. Kelly, "Using a software safety argument pattern catalogue: Two case studies," in *Proc. Intl. Conf. on Comp. Safety, Reliability and Security (SafeComp)*, Sep. 2011.

[10] E. Denney and S. Trac, "A software safety certification tool for automatically generated guidance, navigation and control code," in *IEEE Aerospace Conf. Electronic Proc*, 2008.

[11] FAA, *System Safety Handbook*, Federal Aviation Administration, Dec. 2000.

[12] C. Menon, R. Hawkins, and J. McDermid, "Interim standard of best practice on software in the context of DS 00-56 Issue 4," Soft. Sys. Eng. Initiative, Univ. of York, Standard of Best Practice Issue 1, 2009.

[13] E. Denney, I. Habli, and G. Pai, "Towards measurement of confidence in safety cases," in *Proc. 5th Intl. Symp. Empirical Soft. Eng. and Measurement (ESEM)*, Sep. 2011.

[14] UK Ministry of Defence (MOD), "Safety management requirements for defence systems," Defence Standard 00-56 Issue 4, Jun. 2007.

[15] T. Kelly and J. McDermid, "Safety case patterns – reusing successful arguments," in *Proc. IEE Colloq. on Understanding Patterns and Their Application to Sys. Eng.*, 1998.

[16] P. Bishop, R. Bloomfield, B. Littlewood, A. Povyakalo, and D. Wright, "Towards a formalism for conservative claims about the dependability of software-based systems," *IEEE Trans. Soft. Eng.*, vol. 37, no. 5, pp. 708 – 717, 2011.

[17] V. Basili, G. Caldiera, and D. Rombach, "Goal question metric approach," in *Encyclopedia of Soft. Eng.*. John Wiley, 1994, pp. 528–532.

[18] P. Graydon and J. Knight, "Software process synthesis in assurance based development of dependable systems," in *Proc. European Dependable Comp. Conf. (EDCC)*, Apr. 2010, pp. 75–84.

[19] I. Habli and T. Kelly, "A generic goal-based certification argument for the justification of formal analysis," in *Proc. Certification of Safety-critical Software Controlled Sys. (SafeCert)*, Mar. 2008.

[20] W. Greenwell, J. Knight, C. M. Holloway, and J. Pease, "A taxonomy of fallacies in system safety arguments," in *Proc. Intl. Sys. Safety Conf.*, 2006.

[21] J. Elston, M. Stachura, B. Argrow, E. Frew, and C. Dixon, "Guidelines and Best Practices for FAA Certificate of Authorization Applications for Small Unmanned Aircraft," in *Proc. AIAA Infotech@Aerospace Conf.*, Mar. 2011.