




Software Assurance in an Uncertain World

Marsha Chechik^(✉), Rick Salay, Torin Viger,
Sahar Kokaly, and Mona Rahimi

University of Toronto, Toronto, Canada
`chechik@cs.toronto.edu`

Abstract. From financial services platforms to social networks to vehicle control, software has come to mediate many activities of daily life. Governing bodies and standards organizations have responded to this trend by creating regulations and standards to address issues such as safety, security and privacy. In this environment, the compliance of software development to standards and regulations has emerged as a key requirement. Compliance claims and arguments are often captured in assurance cases, with linked evidence of compliance. Evidence can come from testcases, verification proofs, human judgment, or a combination of these. That is, experts try to build (safety-critical) systems carefully according to well justified methods and articulate these justifications in an assurance case that is ultimately judged by a human. Yet software is deeply rooted in uncertainty; most complex open-world functionality (e.g., perception of the state of the world by a self-driving vehicle), is either not completely specifiable or it is not cost-effective to do so; software systems are often to be placed into uncertain environments, and there can be uncertainties that need to be. We argue that the role of assurance cases is to be the grand unifier for software development, focusing on capturing and managing uncertainty. We discuss three approaches for arguing about safety and security of software under uncertainty, in the absence of fully sound and complete methods: assurance argument rigor, semantic evidence composition and applicability to new kinds of systems, specifically those relying on ML.

1 Introduction

From financial services platforms to social networks to vehicle control, software has come to mediate many activities of daily life. Governing bodies and standards organizations have responded to this trend by creating regulations and standards to address issues such as safety, security and privacy. In this environment, the compliance of software development to standards and regulations has emerged as a key requirement.

Development of safety-critical systems begins with *hazard analysis*, aimed to identify possible causes of harm. It uses severity, probability and controllability of a hazard's occurrence to assign the Safety Integrity Levels (in the automotive industry, these are referred to as ASILs [35]) – the higher the ASIL level,

the more rigor is expected to be put into identifying and mitigating the hazard. Mitigating hazards therefore becomes the main requirement of the system, with system safety requirements being directly linked to the hazards. These requirements are then refined along the LHS of the V until individual modules and their implementation can be built. The RHS includes appropriate testing and validation, used as supporting evidence in developing an argument that the system adequately handles its hazards, with the expectation that the higher the ASIL level, the stronger the required justification of safety is.

Assurance claims and arguments are often captured by *assurance cases*, with linked evidence supporting it. Evidence can come from testcases, verification proofs, human judgment, or a combination of these. Assurance cases organize information allowing argument unfolding in a comprehensive way and ultimately allowing safety engineers to determine whether they trust that the system was adequately designed to avoid systematic faults (before delivery) and adequately detect and react to failures at runtime [35].

Yet software is deeply rooted in uncertainty; most complex open-world functionality (e.g., perception of the state of the world by a self-driving vehicle), is either not completely specifiable or it is not cost-effective to do so [12]. Software systems are often to be placed into uncertain environments [48], and there can be uncertainties that need to be considered at the design phase [20]. Thus, we believe that the role of assurance cases is to *explicitly capture and manage uncertainty coming from different sources, assess it and ultimately reduce it to an acceptable level, either with respect to a standard, company processes, or assessor judgment*. The various software development steps are currently not well integrated, and uncertainty is not expressed or managed explicitly in a uniform manner. Our claim in this paper is that *an assurance case is the unifier among the different software development steps, and can be used to make uncertainties explicit, which also makes them manageable. This provides a well-founded basis for modeling confidence about satisfaction of a critical system quality (security, safety, etc.) in an assurance case, making assurance cases play a crucial role in software development*. Specifically, we enumerate sources of uncertainty in software development. We also argue that organizing software development and analysis activities around the assurance case as a *living document* allows all parts of the software development to explicitly articulate uncertainty, steps taken to manage it, and the degree of confidence that artifacts acting as evidence have been performed correctly. This information can then help potential assessors in checking that the development outcome adequately satisfies the software desired quality (e.g., safety).

The area of system dependability has produced a significant body of work describing how to model assurance cases (e.g., [4, 5, 14, 38]), and how to assess reviewer’s confidence in the argument being made (e.g., [16, 31, 45, 59, 60]). There is also early work on assessing the impact of change on the assurance argument when the system undergoes change [39]. A recent survey [43] provides a comprehensive list of assurance case tools developed over the past 20 years and an analysis of their functionalities including support for assurance case creation,

assessment and maintenance. We believe that the road to truly making assurance cases the grand unifier for software development for complex high-assurance systems has many challenges. One is to be able to successfully argue about safety and security of software under uncertainty, without fully sound and complete methods. For that, we believe that *assurance arguments must be rigorous* and that we need to properly understand how to perform *evidence composition* for traditional systems, but also for *new kinds of systems*, specifically those relying on ML. We discuss these issues below.

Rigor. To be validated or reused, assurance case structures must be as rigorous as possible [51]. Of course, assurance arguments ultimately depend on human judgment (with some facts treated as “obvious” and “generally acceptable”), but the structure of the argument should be fully formal so as to allow to assess its completeness. Bandur and McDermid called this approach “formal modulo engineering expertise” [1].

Evidence Composition. We need to effectively combine the top-down process of uncertainty reduction with the bottom-up process of composing evidence, specifically, evidence obtained from applying testing and verification techniques.

Applicability to “new” kinds of systems. We believe that our view – rigorous, uncertainty-reduction focused and evidence composing – is directly applicable to systems developed using machine learning, e.g., self-driving cars.

This paper is organized as follows: In Sect. 2, we briefly describe syntax of assurance cases. In Sect. 3, we outline possible sources of uncertainty encountered as part of system development. In Sect. 4, we describe the benefits of a rigorous language for assurance cases by way of example. In Sect. 5, we describe, again by way of example, a possible method of composing evidence. In Sect. 6, we develop a high-level assurance case for a pedestrian detection subsystem. We conclude in Sect. 7 with a discussion of possible challenges and opportunities.

2 Background on Assurance Case Modeling Notation

The most commonly used representation for safety cases is the graphical Goal Structuring Notation (GSN) [30], which is intended to support the assurance of critical properties of systems (including safety). GSN is comprised of six core elements – see Fig. 1. Arguments in GSN are typically organized into a tree of the core elements shown in Fig. 1¹. The root is the overall goal to be satisfied by the system, and it is gradually decomposed (possibly via strategies) into sub-goals and finally into solutions, which are the leaves of the safety case. Connections between goals, strategies and solutions represent *supported-by* relations, which indicate inferential or evidential relationships between elements. Goals and strategies may be optionally associated with some contexts, assumptions and/or justifications by means of *in-context-of* relations, which declare a contextual relationship between the connected elements.

¹ In this paper, we use both diamond and triangle shapes interchangeably to depict an “undeveloped” element.



Fig. 1. Core GSN elements from [30].

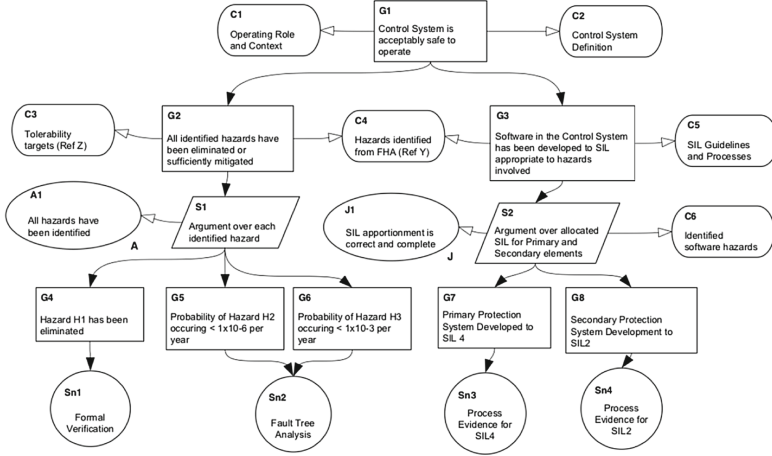


Fig. 2. Example safety case in GSN (from [30]).

For example, consider the safety case in Fig. 2. The overall goal **G1** is that the “Control System is acceptably safe to operate” given its role, context and definition, and it is decomposed into two sub-goals: **G2**, for eliminating and mitigating all identified hazards, and **G3**, for ensuring that the system software is developed to an appropriate ASIL. Assuming that all hazards have been identified, **G2** can in turn be decomposed into three sub-goals by considering each hazard separately (**S1**), and each separate hazard is shown to be satisfied using evidence from formal verification (**Sn1**) or fault tree analysis (**Sn2**). Similarly, under some specific context and justification, **G3** can be decomposed into two sub-goals, each of which is shown to be satisfied by the associated evidence.

3 Sources of Uncertainty in Software Development

In this section, we briefly survey uncertainty in software development, broadly split into the categories of uncertainties about the specifications, about the environment, about the system itself, and about the argument of its safety. For each

part, we aim to address how building an assurance case is related to understanding and mitigating such uncertainties.

Uncertainty in Specifications. Software specifications tend to suffer from incompleteness, inconsistency and ambiguity [42,46]. Specification uncertainty stems from a misunderstanding or an incomplete understanding of how the system is supposed to function in early phases of development; e.g., miscommunication and inability of stakeholders to transfer knowledge due to differing concepts and vocabularies [2,13]; unknown values for sets of known events (a.k.a. the *known unknowns*); and the unknown and unidentifiable events (a.k.a. the *unknown unknowns*) [57].

Recently, machine-learning approaches for interactively learning the software specifications have become popular; we discuss one such example, of pedestrian detection, in Sect. 6. Other mitigations of specification uncertainties, suggested by various standards and research, are identification of edge cases [36], hazard and obstacle analysis [55] to help identify unknown unknowns [35], step-wise refinement to handle partiality in specifications, ontology- [9] and information retrieval-driven requirements engineering approaches [21], as well as generally building arguments about addressing specification uncertainties.

Environmental Uncertainty. The system’s environment can refer to adjacent agents interacting with the system, a human operator using the system, or physical conditions of the environment. Sources of environmental uncertainties have been thoroughly investigated [19,48]. One source originates from unpredictable and changing properties of the environment, e.g., assumptions about actions of other vehicles in the autonomous vehicle domain or assuming that a plane is on the runway if its wheels are turning. Another uncertainty source is input errors from broken sensors, missing, noisy and inaccurate input data, imprecise measurements, or disruptive control signals from adjacent systems. Yet another source might be when changes in the environment affect the specification. For example, consider a robotic arm that moves with the expected precision but the target has moved from its estimated position.

A number of techniques have been developed to mitigate environmental uncertainties, e.g., runtime monitoring systems such as RESIST [10], or machine-learning approaches such as FUSION [18] which self-tune the adaptive behavior of systems to unanticipated changes in the environment. More broadly, environmental uncertainties are mitigated by a careful requirements engineering process, by principled system design and, in assurance cases, by an argument that they had been adequately identified and adequately handled.

System Uncertainties. One important source of uncertainty is faced by developers who do not have sufficient information to make decisions about their system during development. For example, a developer may have insufficient information to choose a particular implementation platform. In [19,48], this source of uncertainty is referred to as *design-time uncertainty*, and some approaches to handling it are offered in [20]. Decisions made while resolving such uncertainties are crucial to put into an assurance argument, to capture the context, i.e.,

a particular platform is selected because of its performance, at the expense of memory requirements.

Another uncertainty refers to correctness of the implementation [7]. This uncertainty lays in the V&V procedure and is caused by whether the implementation of the tool can be trusted, whether the tool is used appropriately (that is, its assumptions are satisfied), and in general, whether a particular verification technique is the right one for verifying the fulfillment of the system requirements [15]. We address some of these uncertainties in Sect. 5.

Argument Uncertainty. The use of safety arguments to demonstrate safety of software-intensive systems raises questions such as the extent to which these arguments can be trusted. That is, how confident are we that a verified, validated software is actually safe? How much evidence and how thorough of an argument do we require for that?

To assess uncertainties which may affect the system’s safety, researchers have proposed techniques to estimate confidence in structured assurance cases, either through qualitative or quantitative approaches [27, 44]. The majority of these are based on the Dempster-Shafer Theory [31, 60], Josang’s Opinion Triangle [17], Bayesian Belief Networks (BBNs) [16, 61], Evidential Reasoning (ER) [45] and weighted averages [59]. The approaches which use BBNs treat safety goals as nodes in the network and try to compute their conditional probability based on given probabilities for the leaf nodes of the network. Dempster-Shafer Theory is similar to BBNs but is based on the *belief function* and its *plausibility* which is used to combine separate pieces of information to calculate the probability. The ER approach [45] allows the assessors to provide individual judgments concerning the trustworthiness and appropriateness of the evidence, building a separate argument from the assurance case.

These approaches focus on assigning and propagating confidence measures but do not specifically address uncertainty in the argument. They also focus on aggregating evidence coming from multiple sources but treat it as a “black box”, instead of how a piece of evidence from one source might compose with another. We look at these questions in Sects. 4 and 5, respectively.

4 Formality in Assurance Cases

As discussed in Sect. 1, we believe that the ultimate goal of an assurance case is to explicitly capture and manage uncertainty, and ultimately reduce it to an acceptable level. Even informal arguments improve safety, e.g., by making people decompose the top level goal case-wise, and examine the decomposed parts critically. But the decomposed cases tend to have an ad hoc structure dictated by experience and preference, with under-explored completeness claims, giving both developers and regulators a false sense of confidence, no matter how confidence is measured, since they feel that their reasoning is rigorous even though it is not [58]. Moreover, as assurance cases are produced and judged by humans, they are typically based on *inductive arguments*. Such arguments are susceptible to fallacies (e.g., arguing through circular reasoning, using justification based

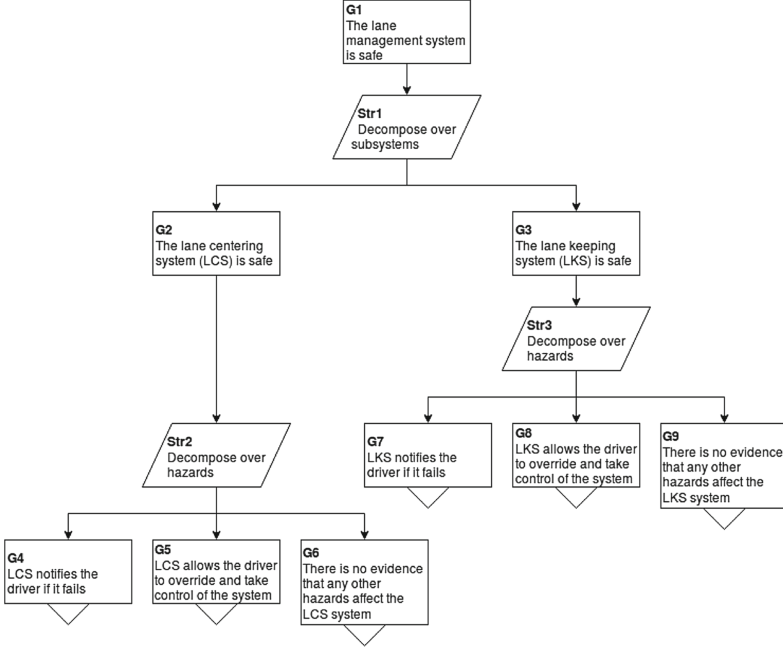


Fig. 3. A fragment of the Lane Management (LMS) Safety case.

on false dichotomies), and evaluations by different reviewers may lead to the discovery of different fallacies [28].

There have been several attempts to improve credibility of an argument by making the argument structure more formal. [25] introduces the notion of confidence maps as an explicit way of reasoning about sources of doubt in an argument, and proposes justifying confidence in assurance arguments through *eliminative induction* (i.e., an argument by eliminating sources of doubt). [29] highlights the need to model both evidential and argumentation uncertainties when evaluating assurance arguments, and considers applications of the formally evaluable extension of Toulmin’s argument style proposed by [56]. [11] details VAA – a method for assessing assurance arguments based on Dempster-Shafer theory. [51] is a proponent of completely deductive reasoning, narrowing the scope of the argument so that it can be formalized and potentially formally checked, using automated theorem provers, arguing that this would give a modular framework for assessing (and, we presume, reusing) assurance cases. [1] relaxes Rushby’s position a bit, aiming instead at formal assurance argumentation “modulo engineering expertise”, and proof obligations about consistency of arguments remain valid even for not fully formal assurance arguments. To this end, they provided a specific formalization of goal validity given validity of subgoals and contexts/context assumptions, resulting in such rules as

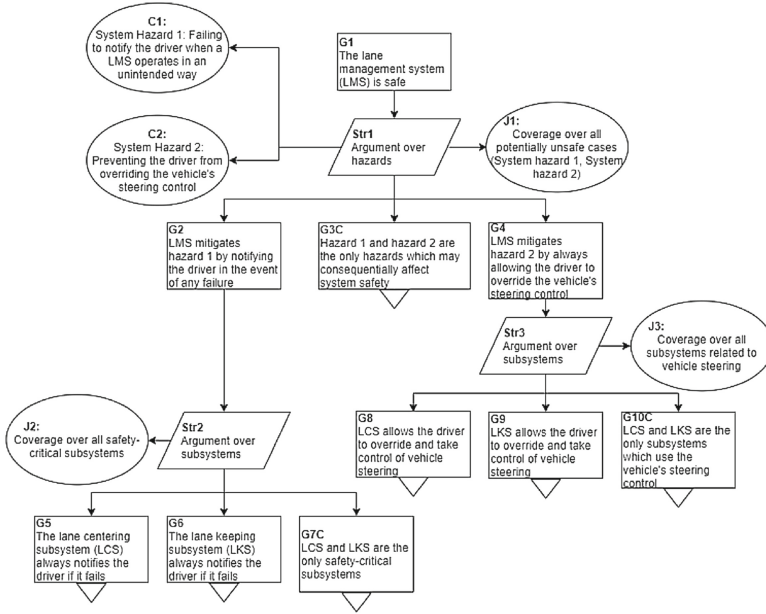


Fig. 4. An alternative representation of the same LMS fragment.

“assumptions on any given element must not be contradictory nor contradict the context assumed for that goal” [1].

Our Position. We believe that a degree of formality in assurance cases can go a long way not only towards establishing its validity, identifying and framing implicit uncertainties and avoiding fallacies, but also supporting assurance case modularity, refactoring and reuse. We illustrate this position on an example.

Example. Consider two partially developed assurance cases that argue that the lane management system (LMS) of a vehicle is safe (Figs. 3 and 4). The top-level safety goal **G1** in Fig. 3 is first decomposed by the strategy **Str1** into a set of subgoals which assert the safety of the LMS subsystems. An assessor can only trust that goals **G2** and **G3** imply **G1** by making an implicit assumption that the system safety is completely determined by the safety of its individual subsystems. Neither the need for this assumption nor the credibility of the assumption itself are made explicit in the assurance case, which weakens the argument and complicates the assessment process. The argument is further weakened by the absence of a completeness claim that all subsystems have been covered by this decomposition.

Strategies **Str2** and **Str3** in Fig. 3 decompose the safety claims about each subsystem into arguments over the relevant hazards. Yet the hazards themselves are never explicitly stated in the assurance case, making the direct relevance of each decomposed goal to its corresponding parent goal, and thus to the argument as a whole, unclear. While goals **G6** and **G9** attempt to provide completeness

claims for their respective decompositions, they do so by citing lack of negative evidence without describing efforts to uncover such evidence. This justification is fallacious and can be categorized as “an argument from ignorance” [28].

Now consider the assurance case in Fig. 4 which presents a variant of the argument in Fig. 3, refined with context nodes, justification nodes and completeness claims. The top-level goal **G1** is decomposed into a set of subgoals asserting that particular hazards have been mitigated, as well as a completeness claim **G3C** stating that hazards **H1** and **H2** are the only ones that may be prevalent enough to defeat claim **G1**. Context nodes **C1** and **C2** define the hazards themselves, which clarifies the relevance of each hazard-mitigating goal. The node **J1** provides a justification for the validity of **Str1** by framing the decomposition as a proof by (exhaustive) cases. That is, **Str1** is justified by the statement that if **H1** and **H2** are the only hazards that could potentially make the system unsafe, then the system is safe if **H1** and **H2** have been adequately mitigated. This rigorous argument can be represented by the logical expression $\mathbf{G3C} \implies ((\mathbf{G2} \wedge \mathbf{G4}) \implies \mathbf{G1})$, and if completeness holds then **G2** and **G4** are sufficient to show **G1**. We now have a rigorous argument step that our confidence in **G1** is a direct consequence of confidence in its decomposed goals **G2**, **G3C** and **G4**, even though there may still be uncertainty in the evidential evaluation of **G2**, **G3C** and **G4**. That is, uncertainty has been made explicit and can be reasoned about at the evidential level. By removing argumentation uncertainty and explicating implicit assumptions, we get a more comprehensive framework for assurance case evaluation, where the relation between all reasoning steps is formally clear. Note that if the justification provides an inference rule, then the argument becomes deductive. Otherwise, it is weaker (the justification node can be used to quantify just *how* weaker) but still rigorous.

While the completeness claim **G3C** in Fig. 4 may be directly supported by evidence, the goals **G2** and **G4** are further decomposed by the strategies **Str2** and **Str3**, respectively, which represent decompositions over subsystems. These strategies are structured similarly to **Str1**, and can be expressed by the logical expressions $\mathbf{G7C} \implies ((\mathbf{G5} \wedge \mathbf{G6}) \implies \mathbf{G2})$ and $\mathbf{G10C} \implies ((\mathbf{G8} \wedge \mathbf{G9}) \implies \mathbf{G4})$, respectively. In Fig. 3, a decomposition by subsystems was applied directly to the top-level safety goal which necessitated a completeness claim that the safety of all individual subsystems implied safety of the entire system. Instead, the argument in Fig. 4 only needs to show that the set of subsystems in each decomposition is complete w.r.t. a particular hazard, which may be a more feasible claim to argue. This ability to transform an argument into a more easily justifiable form is another benefit of arguing via rigorous reasoning steps.

5 Combining Evidence

Evidence for assurance cases can come from a variety of sources: results from different testing and verification techniques, human judgment, or their combination. Multiple testing and verification techniques may be used to make the evidence more complete. A verification technique *complements* another if it is able

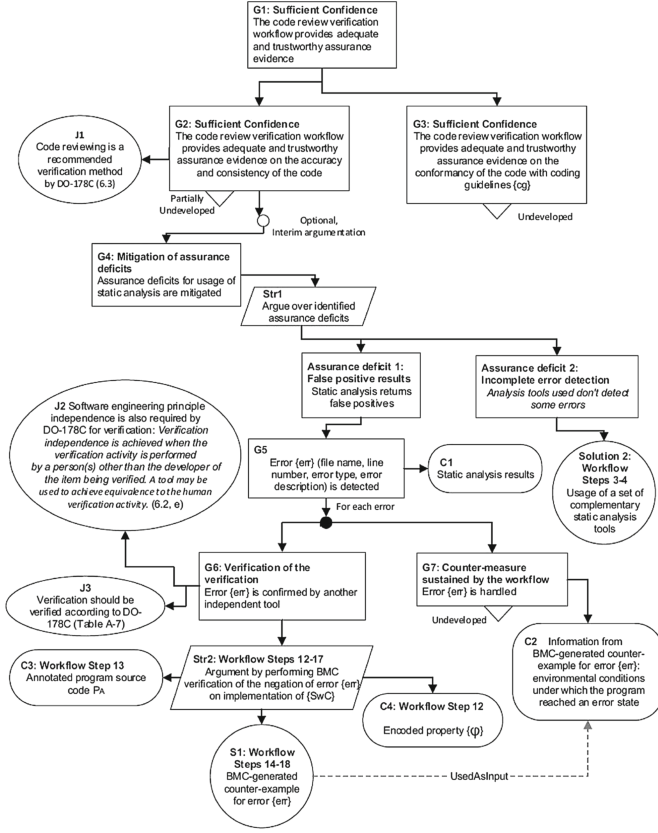


Fig. 5. Confidence argument for code review workflow (from [6]).

to verify types of requirements which cannot be verified by the other technique. For example, results of verification of properties via a bounded model checker (BMC) are complemented by additional test cases [8]. A verification technique *supports* another if it is used to detect faults in the other's verification results, thus providing backing evidence [33]. For example, a model checking technique may support a static analysis technique by verifying the faults detected [6]. Note that these approaches are principally different from just aggregating evidence treating it as a blackbox!

Habli and Kelly [32] and Denney and Pai [15] present safety case patterns for the use of formal method results for certification. Bennion et al. [3] present a safety case for arguing the compliance of a particular model checker, namely, the Simulink Design Verifier for DO-178C. Gallina and Andrews [23] argue about adequacy of a model-based testing process, and Carlan et al. [7] provide a safety pattern for choosing and composing verification techniques based on how they

contribute to the identification or mitigation of systematic faults known to affect system safety.

Our Position. We, as a community, need to figure out the precise conditions under which particular testing and verification techniques “work” (e.g., modeling floating-point numbers as reals, making a small model hypothesis to justify sufficiency of a particular loop unrolling, etc.), and how they are intended to be composed in order to reduce uncertainty about whether software satisfies its specification. We illustrate a particular composition here.

Example. In this example, taken from [6], a model checker supports static analysis tools (that produce false negatives) by verifying the detected faults [6]. The assurance case is based on a workflow (not shown here) where an initial review report is constructed, by running static analysis tools and possibly peer code reviews. Then the program is annotated with the negation of each potential erroneous behavior as a desirable property for the program, and given to a model-checker. If the model-checker is able to verify the property, it is removed from the initial review report and not considered as an error. If the model-checker finds a violation, the alleged error is confirmed. In this case, a weakest-precondition generation mechanism is applied to find out the environmental conditions (external parameters that are not under the control of the program) under which the program shows the erroneous behavior. These conditions and the error trace are then added to the error description.

The paper [6] presents both the assurance case and the confidence argument for the code review workflow. We reproduce only the latter here (see Fig. 5), focusing on reducing uncertainty about the accuracy and consistency of the code property (goal **G2**). False positives generated by static analysis are mitigated using BMC – a method with a completely different verification rationale, thus implementing the safety engineering principle of independence (**J2**). Strategy (**Str2**) explains how errors can be confirmed or dismissed using BMC (goal **G6**). The additional information given by BMC can be used for the mitigation of the error (**C2**).

This approach takes good steps towards mitigating particular assurance deficits using a composition of verification techniques but leaves open several problems: how to ensure that BMC runs under the same environmental conditions as the static analysis tools? how deeply should the loops be unrolled? what to do with cases when the model-checker runs out of resources without giving a conclusive answer? and in general, what are the conditions under which it is safe to trust the “yes” answers of the model-checker.

6 Assurance Cases for ML Systems

Academia and industry are actively building systems using AI and machine learning, including a rapid push for ML in safety-critical domains such as medical devices and self-driving cars. For their successful adoption in society, we need to ensure that they are trustworthy, including obtaining confidence in their behavior and robustness.

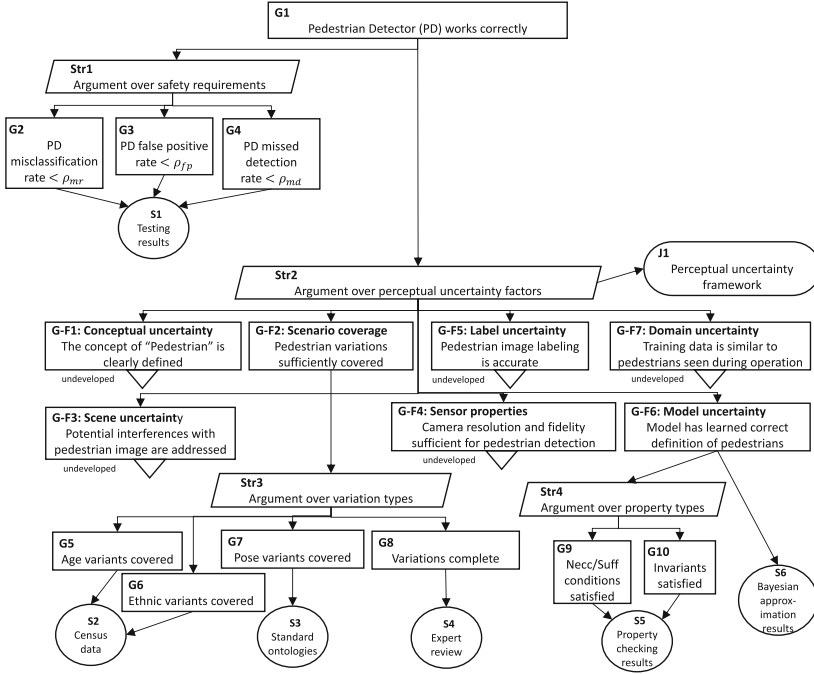


Fig. 6. A partially developed GSN safety case of pedestrian detector example.

Significant strides have already been made in this space, from extending mature testing and verification techniques to reasoning about neural networks [24, 37, 47, 54] for properties such as safety, robustness and adequate handling of adversarial examples [26, 34]. There is active work in designing systems that balance learning under uncertainty and acting safely, e.g., [52] as well as the broad notion of fairness and explainability in AI, e.g., [49].

Our Position. We believe that assurance cases remain a unifying view for ML-based systems just as much as for more conventional systems, allowing us to understand how the individual approaches fit into the overall goal of assuring safety and reliability and where there are gaps.

Example. We illustrate this idea with an example of a simple pedestrian detector (PD) component used as part of an autonomous driving system. The functions that PD supports consist of detection of objects in the environment ahead of the vehicle, classification of an object as a *pedestrian* or *other*, and localization of the position and extent of the pedestrian (indicated by bounding box). We assume that PD is implemented as a convolutional deep neural network with various stages to perform feature extraction, proposing regions containing objects and classification of the proposed objects. This is a typical approach for two-stage object detectors (e.g., see [50]).

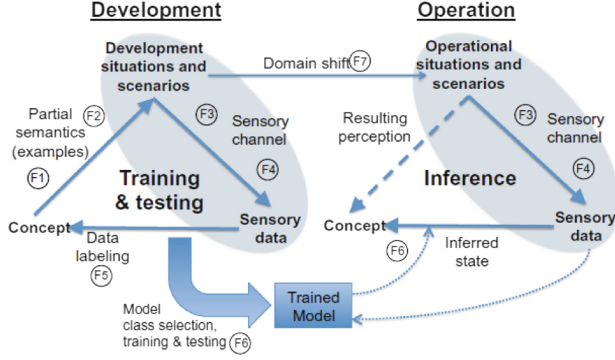


Fig. 7. A framework for factors affecting perceptual uncertainty (source: [12]).

As part of a safety critical system, PD contributes to the satisfaction of a top-level safety goal requiring that the vehicle always maintain a safe distance from all pedestrians. Specific safety requirements for PD can be derived from this goal, such as (RQ1) PD misclassification rate (i.e., classifying a pedestrian as “other”) must be less than ρ_{mc} , (RQ2) PD false positive rate (i.e., classifying any non-pedestrian object or non-object as “pedestrian”) must be less than ρ_{fp} , and (RQ3) PD missed detection rate (i.e., missing the presence of pedestrian) must be less than ρ_{md} . Here, the parameters ρ_{mc} , ρ_{fp} and ρ_{md} must be derived in conjunction with the control system that uses the output from PD to plan the vehicle trajectory.

The partially developed safety case for PD is shown in Fig. 6. The three safety requirements are addressed via the strategy **Str1** and, as expected, testing results are given as evidence of their satisfaction. However, since testing can only provide limited assurance about the behaviour of PD in operation, we use an additional strategy, **Str2**, to argue that a rigorous method was followed to develop PD. Specifically, we follow the framework of [12] for identifying the factors that lead to uncertainty in ML-based perceptual software such as PD.

The framework is defined at a high level in Fig. 7. The left “perception triangle” shows how the perceptual concept (in the case of PD, the concept “pedestrian”) can occur in various scenarios in the world, how it is detected using sensors such as cameras, and how this can be used to collect and label examples in order to train an ML component to learn the concept. The perception triangle on the right is similar but shows how the trained ML component can be used during the system operation to make inferences (e.g., perform the pedestrian detection). The framework identifies seven factors that could contribute to uncertainty in the behaviour of the perceptual component. A safety case demonstrating a rigorous development process should provide evidence that each factor has been addressed.

In Fig. 6, strategy **Str2** uses the framework to argue that the seven factors are adequately addressed for PD. We illustrate development of two of these factors

here. Scenario coverage (Goal **G-F2**) deals with the fact that the training data must represent the concept in a sufficient variety of scenarios in which it could occur in order for the training to be effective. The argument here first decomposes this goal into different types of variation (**Str3**) and provides appropriate evidence for each. The adequacy of age and ethnicity variation in the data set is supported by census data (**S2**) about the range of these dimensions of variation in the population. The variation in the pedestrian pose (i.e., standing, leaning, crouching, etc.) is supplied by a standard ontology of human postures (**S3**). Finally, evidence that the types are adequate to provide sufficient coverage of variation (completeness) is provided by an expert review (**S4**).

Another contributing factor developed in Fig. 6 is model uncertainty (Goal **G-F6**). Since there is only finite training data, there can be many possible models that are equally consistent with the training data, and the training process could produce any one of them, i.e., there is residual uncertainty whether the produced model is in fact correct. The presence of model uncertainty means that while the trained model may perform well on inputs similar to the training data, there is no guarantee that it will produce the right output for other inputs. Some evidence of good behaviour here can be gathered if there are known properties that partially characterize the concept and can be checked. For example, a reasonable necessary condition for PD is that the object being classified as a pedestrian should be less than 9ft tall. Another useful property type is an invariant, e.g., a rotated pedestrian image is still a pedestrian. Tools for property checking of neural networks (e.g., [37]) can provide this kind of evidence (**S5**). Another way to deal with model uncertainty is to estimate it directly. Bayesian deep learning approaches [22] can do this by measuring the degree of disagreement between multiple trained models that are equally consistent with the training data. The more the models are in agreement are about how to classify a new input, the less model uncertainty is present and the more confident one can be in the prediction. Using this approach on a test data set can provide evidence (**S6**) about the degree of model uncertainty in the model. This approach can also be used during the operation to generate a confidence score in each prediction and use a fault tolerance strategy that takes a conservative action when the confidence falls below a threshold.

7 Summary and Future Outlook

In this paper, we tried to argue that an assurance case view on establishing system correctness provides a way to unify different components of the software development process and to explicitly manage uncertainty. Furthermore, although our examples came from the world of safety-critical automotive systems, the assurance case view is broadly applicable to a variety of systems, not just those in the safety-critical domain and includes those constructed by non-traditional means such as ML. This view is especially relevant to much of the research activity being conducted by the ETAPS community since it allows, in principle, to understand how each method contributes to the overall problem of system assurance.

Most traditional assurance methods aim to build an informal argument, ultimately judged by a human. However, while these are useful for showing compliance to standards and are relatively easy to construct and read, such arguments may not be rigorous, missing essential properties such as completeness, independence, relevance, or a clear statement of assumptions [51]. As a result, fallacies in existing assurance cases are present in abundance [28]. To address this weakness, we argued that building assurance cases should adhere to systematic principles that ensure rigor. Of course, not all arguments can be fully deductive since relevance and admissibility of evidence is often based on human judgment. Yet, an explicit modeling and management of uncertainty in evidence, specifications and, assumptions as well as the clear justification of each step can go a long way toward making such arguments valid, reusable, and generally useful in helping produce high quality software systems.

Challenges and Opportunities. Achieving this vision has a number of challenges and opportunities. In our work on impact assessment of model change on assurance cases [39, 40], we note that even small changes to the system may have significant impact on the assurance case. Because creation of an assurance case is costly, this brittleness must be addressed. One opportunity here is to recognize that assurance cases can be refactored to improve their qualities without affecting their semantics. For example, in Sect. 4, we showed that the LMS safety claim could either be decomposed first by hazards and then by subsystems or vice versa. Thus, we may want to choose the order of decomposition based on other goals, e.g., to minimize the impact of change on the assurance case by pushing the affected subgoals lower in the tree. Another issue is that complex systems yield correspondingly complex assurance cases. Since these must ultimately be judged by humans, we must manage the cognitive load the assurance case puts on the assessor. This creates opportunities for mechanized support, both in terms of querying, navigating and analyzing assurance cases as well as in terms of modularization and reuse of assurance cases.

Evidence composition discussed in Sect. 5 also presents significant challenges. While standards such as DO-178C and ISO26262 give recommendations on the use of testing and verification, it is not clear how to compose partial evidence or how to use results of one analysis to support another. Focusing on how each technique reduces potential faults in the program, clearly documenting their context of applicability (e.g., the small model hypothesis justifying partial unrolling of loops, properties not affected by approximations of complex program operations and datatypes often done by model-checkers, connections between the modeled and the actual environment, etc.) and ultimately connecting them to reducing uncertainties about whether the system satisfies the essential property are keys to making tangible progress in this area.

Finally, in Sect. 6, we showed how the assurance case view could apply to new development approaches such as ML. Although such new approaches provide benefits over traditional software development, they also create challenges for assurance. One challenge is that analysis techniques used for verification may be immature. For example, while neural networks have been studied since the

1950's, pragmatic approaches to their verification have been investigated only recently [53]. Another issue is that prerequisites for assurance may not be met by the development approach. For example, although they are expressive, neural networks suffer from uninterpretability [41] – that is, it is not feasible for a human to examine a trained network and understand what it is doing. This is a serious obstacle to assurance because formal and automated methods account for only part of the verification process, augmented by reviews. As a result, increasing the interpretability of ML models is an active area of current research.

While all these challenges are significant, the benefit of addressing them is worth the effort. As our world moves towards increasing automation, we must develop approaches for assuring the dependability of the complex systems we build. Without this, we either stall progress or run the risk of endangering ourselves – neither alternative seems desirable.

References

1. Bandur, V., McDermid, J.: Informing assurance case review through a formal interpretation of GSN core logic. In: Koornneef, F., van Gulijk, C. (eds.) SAFECOMP 2015. LNCS, vol. 9338, pp. 3–14. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24249-1_1
2. Bell, T.E., Thayer, T.A.: Software requirements: are they really a problem? In: Proceedings of the 2nd International Conference on Software Engineering, pp. 61–68. IEEE Computer Society Press (1976)
3. Bennion, M., Habli, I.: A candid industrial evaluation of formal software verification using model checking. In: Companion Proceedings of ICSE 2014, pp. 175–184 (2014)
4. Bloomfield, R., Bishop, P.: Safety and assurance cases: past, present and possible future - an Adelard perspective. In: Dale, C., Anderson, T. (eds.) Safety-Critical Systems: Problems, Process and Practice, pp. 51–67. Springer, London (2010). https://doi.org/10.1007/978-1-84996-086-1_4
5. Brunel, J., Cazin, J.: Formal verification of a safety argumentation and application to a complex UAV system. In: Ortmeier, F., Daniel, P. (eds.) SAFECOMP 2012. LNCS, vol. 7613, pp. 307–318. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33675-1_27
6. Carlan, C., Beyene, T.A., Ruess, H.: Integrated formal methods for constructing assurance cases. In: Proceedings of ISSRE 2016 Workshops (2016)
7. Cărlan, C., Gallina, B., Kacianka, S., Breu, R.: Arguing on software-level verification techniques appropriateness. In: Tonetta, S., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2017. LNCS, vol. 10488, pp. 39–54. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66266-4_3
8. Cărlan, C., Ratiu, D., Schätz, B.: On using results of code-level bounded model checking in assurance cases. In: Skavhaug, A., Guiochet, J., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2016. LNCS, vol. 9923, pp. 30–42. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45480-1_3
9. Castaameda, V., Ballejos, L., Caliusco, M.L., Galli, M.R.: The use of ontologies in requirements engineering. Glob. J. Res. Eng. **10**(6) (2010)
10. Cooray, D., Malek, S., Roshandel, R., Kilgore, D.: RESISTing reliability degradation through proactive reconfiguration. In: Proceedings of ASE 2010, pp. 83–92. ACM (2010)

11. Cyra, L., Gorski, J.: Support for argument structures review and assessment. *J. Reliab. Eng. Syst. Saf.* **96**, 26–37 (2011)
12. Czarnecki, K., Salay, R.: Towards a framework to manage perceptual uncertainty for safe automated driving. In: Gallina, B., Skavhaug, A., Schoitsch, E., Bitsch, F. (eds.) *SAFECOMP 2018*. LNCS, vol. 11094, pp. 439–445. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99229-7_37
13. Davis, A., et al.: Identifying and measuring quality in a software requirements specification. In: *1993 Proceedings First International Software Metrics Symposium*, pp. 141–152. IEEE (1993)
14. de la Vara, J.L.: Current and necessary insights into SACM: an analysis based on past publications. In: *Proceedings of RELAW 2014*, pp. 10–13. IEEE (2014)
15. Denney, E., Pai, G.: Evidence arguments for using formal methods in software verification. In: *Proceedings of ISSRE 2013 Workshops* (2013)
16. Denney, E., Pai, G., Habli, I.: Towards measurement of confidence in safety cases. In: *Proceedings of ESEM 2011* (2011)
17. Duan, L., Rayadurgam, S., Heimdahl, M.P.E., Sokolsky, O., Lee, I.: Representing confidence in assurance case evidence. In: Koornneef, F., van Gulijk, C. (eds.) *SAFECOMP 2015*. LNCS, vol. 9338, pp. 15–26. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24249-1_2
18. Elkhodary, A., Esfahani, N., Malek, S.: FUSION: a framework for engineering self-tuning self-adaptive software systems. In: *Proceedings of FSE 2010*, pp. 7–16. ACM (2010)
19. Esfahani, N., Malek, S.: Uncertainty in self-adaptive software systems. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) *Software Engineering for Self-Adaptive Systems II*. LNCS, vol. 7475, pp. 214–238. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35813-5_9
20. Famelis, M., Chechik, M.: Managing design-time uncertainty. *J. Softw. Syst. Model.* (2017)
21. Fanmuy, G., Fraga, A., Llorens, J.: Requirements verification in the industry. In: Hammami, O., Krob, D., Voirin, J.L. (eds.) *Complex Systems Design & Management*, pp. 145–160. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-25203-7_10
22. Gal, Y., Ghahramani, Z.: Dropout as a Bayesian approximation: representing model uncertainty in deep learning. In: *Proceedings of ICML 2016*, pp. 1050–1059 (2016)
23. Gallina, B., Andrews, A.: Deriving verification-related means of compliance for a model-based testing process. In: *Proceedings of DASC 2016* (2016)
24. Gehr, T., Milman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: AI2: safety and robustness certification of neural networks with abstract interpretation. In: *Proceedings of IEEE S&P 2018* (2018)
25. Goodenough, J., Weinstock, C., Klein, A.: Eliminative induction: a basis for arguing system confidence. In: *Proceedings of ICSE 2013* (2013)
26. Gopinath, D., Wang, K., Zhang, M., Pasareanu, C., Khunshid, S.: Symbolic execution for deep neural networks. [arXiv:1807.10439v1](https://arxiv.org/abs/1807.10439v1) (2018)
27. Graydon, P.J., Holloway, C.M.: An investigation of proposed techniques for quantifying confidence in assurance arguments. *J. Saf. Sci.* **92**, 53–65 (2017)
28. Greenwell, W.S., Knight, J.C., Holloway, C.M., Pease, J.J.: A taxonomy of fallacies in system safety arguments. In: *Proceedings of ISSC 2006* (2006)
29. Grigorova, S., Maibaum, T.: Argument evaluation in the context of assurance case confidence modeling. In: *Proceedings of ISSRE Workshops* (2014)

30. GSN: Goal Structuring Notation Working Group, “GSN Community Standard Version 1”, November 2011. <http://www.goalstructuringnotation.info/>
31. Guiochet, J., Hoang, Q.A.D., Kaaniche, M.: A model for safety case confidence assessment. In: Koornneef, F., van Gulijk, C. (eds.) SAFECOMP 2015. LNCS, vol. 9337, pp. 313–327. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24255-2_23
32. Habli, I., Kelly, T.: A generic goal-based certification argument for the justification of formal analysis. ENTCS **238**(4), 27–39 (2009)
33. Hawkins, R., Kelly, T.: A structured approach to selecting and justifying software safety evidence. In: Proceedings of SAFECOMP 2010 (2010)
34. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Majumdar, R., Kunčák, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 3–29. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_1
35. International Organization for Standardization: ISO 26262: Road Vehicles – Functional Safety, 1st version (2011)
36. International Organization for Standardization: ISO/AWI PAS 21448: Road Vehicles – Safety of the Intended Functionality (2019)
37. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčák, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5
38. Kelly, T., Weaver, R.: The goal structuring notation – a safety argument notation. In: Proceedings of Dependable Systems and Networks Workshop on Assurance Cases (2004)
39. Kokaly, S., Salay, R., Cassano, V., Maibaum, T., Chechik, M.: A model management approach for assurance case reuse due to system evolution. In: Proceedings of MODELS 2016, pp. 196–206. ACM (2016)
40. Kokaly, S., Salay, R., Chechik, M., Lawford, M., Maibaum, T.: Safety case impact assessment in automotive software systems: an improved model-based approach. In: Tonetta, S., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2017. LNCS, vol. 10488, pp. 69–85. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66266-4_5
41. Lipton, Z.C.: The mythos of model interpretability. Commun. ACM **61**(10), 36–43 (2018)
42. Lutz, R.R.: Analyzing software requirements errors in safety-critical, embedded systems. In: Proceedings of IEEE International Symposium on Requirements Engineering, pp. 126–133. IEEE (1993)
43. Maksimov, M., Fung, N.L.S., Kokaly, S., Chechik, M.: Two decades of assurance case tools: a survey. In: Gallina, B., Skavhaug, A., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2018. LNCS, vol. 11094, pp. 49–59. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99229-7_6
44. Nair, S., de la Vara, J.L., Sabetzadeh, M., Falessic, D.: Evidence management for compliance of critical systems with safety standards: a survey on the state of practice. Inf. Softw. Technol. **60**, 1–15 (2015)
45. Nair, S., Walkinshaw, N., Kelly, T., de la Vara, J.L.: An evidential reasoning approach for assessing confidence in safety evidence. In: Proceedings of ISSRE 2015 (2015)
46. Nikora, A., Hayes, J., Holbrook, E.: Experiments in automated identification of ambiguous natural-language requirements. In: Proceedings 21st IEEE International Symposium on Software Reliability Engineering. IEEE Computer Society, San Jose (2010, to appear)

47. Pei, K., Cao, Y., Yang, J., Jana, S.: DeepXplore: automated whitebox testing of deep learning systems. In: Proceedings of SOSP 2017 (2017)
48. Ramirez, A.J., Jensen, A.C., Cheng, B.H.: A taxonomy of uncertainty for dynamically adaptive systems. In: Proceedings of SEAMS 2012 (2012)
49. Ras, G., van Gerven, M., Haselager, P.: Explanation methods in deep learning: users, values, concerns and challenges. In: Escalante, H.J., et al. (eds.) *Explainable and Interpretable Models in Computer Vision and Machine Learning*. TSSCML, pp. 19–36. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98131-4_2
50. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. In: *Advances in Neural Information Processing Systems*, pp. 91–99 (2015)
51. Rushby, J., Xu, X., Rangarajan, M., Weaver, T.L.: Understanding and evaluating assurance cases. Technical report CR-2015-218802, NASA (2015)
52. Sadigh, D., Kapoor, A.: Safe control under uncertainty with probabilistic signal temporal logic. In: Proceedings of RSS 2016 (2016)
53. Seshia, S.A., Sadigh, D.: Towards verified artificial intelligence. CoRR, abs/1606.08514 (2016)
54. Tian, Y., Pei, K., Jana, S., Ray, B.: DeepTest: automated testing of deep-neural-network-driven autonomous cars. In: Proceedings of ICSE 2018 (2018)
55. Van Lamsweerde, A.: Goal-oriented requirements engineering: a guided tour. In: Proceedings of RE 2001, pp. 249–262. IEEE (2001)
56. Verheij, B.: Evaluating arguments based on Toulmin’s scheme. *Argumentation* **19**(3), 347–371 (2005)
57. Ward, S., Chapman, C.: Transforming project risk management into project uncertainty management. *Int. J. Proj. Manag.* **21**(2), 97–105 (2003)
58. Wassyng, A.: Private Communication (2019)
59. Yamamoto, S.: Assuring security through attribute GSN. In: Proceedings of ICITCS 2015 (2015)
60. Zeng, F., Lu, M., Zhong, D.: Using DS evidence theory to evaluation of confidence in safety case. *J. Theoret. Appl. Inf. Technol.* **47**(1) (2013)
61. Zhao, X., Zhang, D., Lu, M., Zeng, F.: A new approach to assessment of confidence in assurance cases. In: Ortmeier, F., Daniel, P. (eds.) *SAFECOMP 2012*. LNCS, vol. 7613, pp. 79–91. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33675-1_7

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter’s Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter’s Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

