Assurance Case Development as Data: A Manifesto

Claudio Menghi*, Torin Viger†, Alessio Di Sandro†, Chris Rees‡, Jeff Joyce‡ and Marsha Chechik†

* McMaster University, Hamilton, Canada
Email: menghic@mcmaster.ca
‡University of Toronto, Toronto, Canada
Email: torin.viger@mail.utoronto.ca, adisandro@cs.toronto.edu, chechik@cs.toronto.edu
§Critical Systems Labs, Vancouver, Canada
Email: {chris.rees,jeff.joyce}@cslabs.com

***Abstract*—Safety problems can be costly and catastrophic. Engineers typically rely on assurance cases to ensure their systems are adequately safe. Building safe software systems requires engineers to iteratively design, analyze and refine assurance cases until sufficient safety evidence is identified. The assurance case development is typically manual, time-consuming, and far from being straightforward. This paper presents a *manifesto* for our forward-looking idea: using *assurance cases as data*. We argue that engineers produce a lot of data during the assurance case development process, and such data can be collected and used to effectively improve this process. Therefore, in this manifesto, we propose to monitor the assurance case development activities, treat assurance cases as data, and learn suggestions that help safety engineers in designing safer systems.**

***Index Terms*—Safety, Assurance Cases, Data Analysis, Recommendations**

I. Introduction

Software safety problems can be costly and catastrophic. For example, automotive software failures have led to the loss of human lives and required automotive companies to recall their vehicles at significant costs [1] to them. These problems are likely to increase since the software is "eating the car" [2].

To prevent safety problems, engineers extensively *analyze the safety* of their systems [3] to ensure mitigation of safety hazards and compliance with safety standards (e.g., ISO 26262 [4]). Safety analysis involves multiple parties, including software engineers, safety engineers, and regulators, and is supported by industrial tools, such as SOCRATES [5], MEDINI [6], and ASTAH [7]. *Assurance cases* (ACs) are structured safety arguments supported by evidence designed to show that a system is sufficiently safe.

A number of academic and industrial tools have been developed to provide *automated support* for safety design activities, as described in recent surveys [8], [9], [10], [11], [12], [13], [3]. These tools provide support for many activities including creation, maintenance [10], assessment [9], collaboration, reporting and integration [8]. Despite the automated tool support, the design of the safety argument is still mostly manual, time-consuming, and error-prone [3], [10]. Therefore, as also argued by recent surveys (e.g., [12]), engineers need techniques to develop ACs more effectively.

This paper presents a manifesto for treating *assurance cases as data*. This manifesto declares our intentions, motives, and views [14]. Inspired by the works on code recommendation, our forward-looking idea is to monitor safety designers during the AC development, collect and analyze data to provide (helpful) recommendations for the safety argument. Safety design data is collected by recording the safety design activities; analysis can rely on existing recommendation algorithms; finally, recommendations are shown to safety designers.

We believe the ideas presented by this manifesto are likely to *significantly impact* software engineering practice. First, safety recommendations may enable software engineers to detect safety relations between the nodes of the ACs that are difficult to identify manually without proper automated support. Second, safety recommendations will likely reduce the development costs: they could reduce the time required to develop ACs by exposing safety relations between the AC nodes that require time to be manually identified.

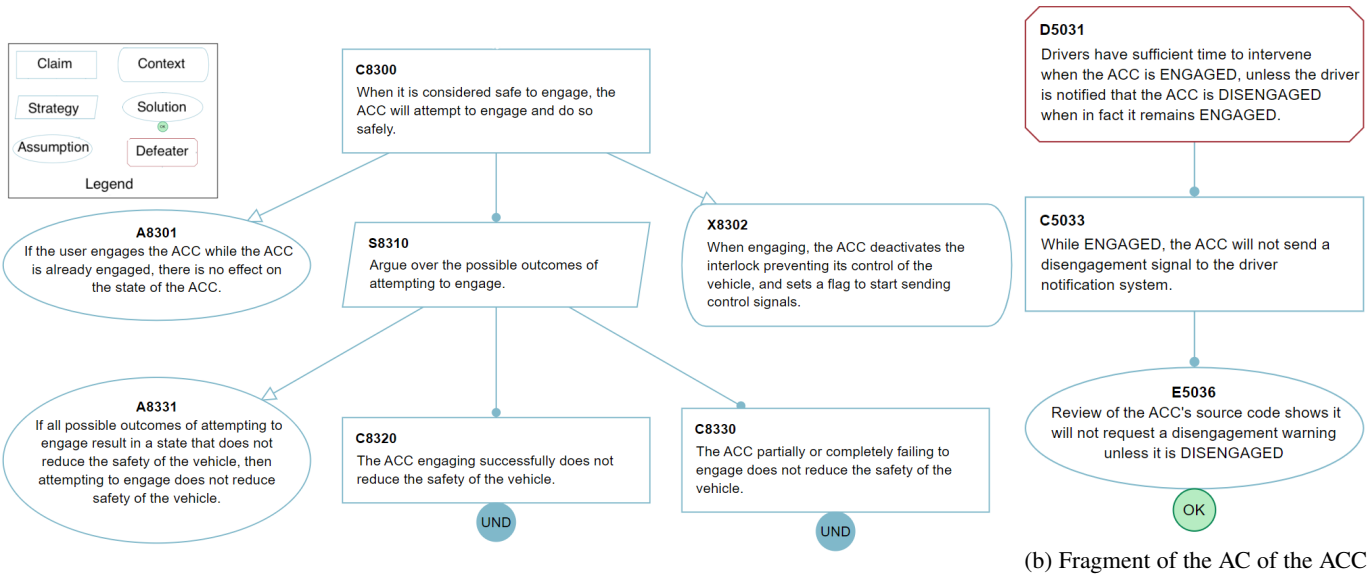
We are not aware of any prior work that proposes considering AC development as data. While our ideas are inspired by previous work on code recommendations, our usage scenario and final goals are significantly different. Unlike source code development, safety analysis activities require software engineers to iteratively design, analyze and refine ACs until sufficient evidence is identified to claim the system safety. This goal makes the context of usage, type, and amount of data significantly different from those in code recommendations.

This paper is organized as follows. Sec. II presents our motivation through a running example. Sec. III presents our intentions. Sec. IV presents our views on the expected results. Sec. V presents related work. Sec. VI concludes the paper.

II. Motivation

This section motivates our work and introduces our running example, an *adaptive cruise control* (ACC) system for a vehicle developed by our industrial partner Critical Systems Labs (CSL) [15].

ACs are used by safety engineers to analyze vehicle safety, and are commonly constructed using Goal Structuring Notation (GSN) [16]. GSN ACs are tree-like argument structures designed to show that a system is safe with respect to its identified hazards. They start with a high-level claim about system safety, and iteratively decompose this claim into more refined subclaims that can be directly supported by evidence.



(a) Fragment of the AC referring to the safe engaging of the ACC.

(b) Fragment of the AC of the ACC referring to the driver behavior.

Fig. 1: Two fragments of an AC fragment for a vehicle's adaptive cruise control system.

Figures 1a and 1b present two fragments of an AC for the ACC modeled with a variation of GSN called *Eliminative Argumentation* (EA) [17] that supports *defeaters*, i.e., nodes giving explicit reasons to doubt safety claims [18]. The ACs in Figures 1a and 1b are fragments of a larger AC for the ACC system containing 315 GSN nodes. GSN nodes represent claims, strategies, solutions, contexts, assumptions, justifications, and defeaters. As ACs are trees, each node is the child of another node (except the tree root).

- *Claims* are assertions that specify that a system satisfies a safety property. For example, the claim *C8320* of Figure 1a asserts that "the ACC engaging successfully does not reduce the safety of the vehicle".
- *Strategies* are justifications for how a claim has been addressed through decomposition of the argument. For example, strategy *S8310* decomposes claim *C8300* into claims *C8320* and *C8330* by arguing that every possible outcome is safe when the system attempts to engage.
- *Assumptions* document conditions that are assumed to be true in the system's operational environment. For example, the assumption node *A8301* specifies that "if the user engages the ACC while the ACC is already engaged, there is no effect on the state of the ACC".
- *Contexts* provide contextual information or constrain the scope over which a claim is made. For example, the context node *X8302* specifies that "when engaging, the ACC deactivates the interlock preventing its control of the vehicle, and sets a flag to start sending control signals".
- *Solutions* provide evidence to show that a claim is satisfied. For example, *E5036* in Figure 1b supports claim *C5033* with evidence from source code analysis.
- *Defeaters* are doubts representing ways in which a safety claim may be defeated and are usually resolved by showing that the defeater is mitigated. For example, *D5031* in Figure 1b asserts that *C5030* may be defeated if a driver is falsely notified that the ACC is disengaged.

AC development is an *incremental* process in which ACs are iteratively revised until sufficient evidence about the system safety is collected. During the development of an AC, safety engineers add, remove, and modify claims, strategies, solutions, contexts, assumptions, justifications, and defeaters. For example, consider claim *C5033* in Figure 1b which asserts that disengagement signals will never be sent by the ACC while the system is engaged. During the safety analysis process, a safety engineer may decide that the ACC should instead send disengagement signals to the driver notification system several seconds before disengaging (i.e., while still engaged) so that the driver has sufficient time to react and take control of the vehicle. To reflect this change, they may modify claim *C5033* to say "While ENGAGED, the ACC will only send a disengagement signal to the driver notification system within three seconds of disengaging", and modify its corresponding evidence node (*E5036*) to say "Review of the ACC's source code shows it will only request a disengagement warning within three seconds of disengaging". After this change is applied, assessing the *impact* of the change on the safety argument is difficult, time-consuming, and error-prone. For example, safety engineers may erroneously consider only the fragment of the AC in Figure 1b for safety analysis; however, this change may also impact context node *X8302* in Figure 1a. This node asserts that the ACC sends appropriate signals while it is engaging, and may also need to be changed to specify that signals will be sent several seconds before engaging to give the driver sufficient time to react. However, the large number of AC elements may prevent engineers from realizing that this element also needs to be updated after they make a change.

Safety engineers need automated tools that record safety

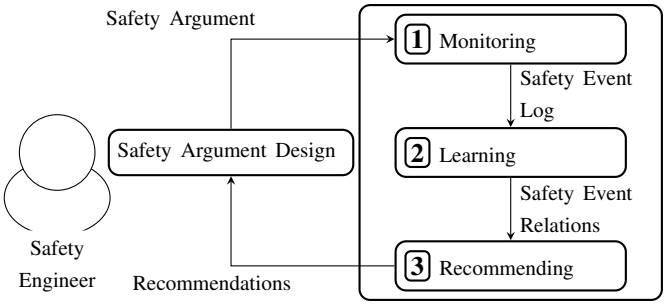


Fig. 2: Assurance case development as data: overview.

design activities and collect data and that, after each change, can effectively and efficiently provide recommendations by relying on these data. For example, after a change to the claim *C5033* of Figure 1b, the recommendation framework should suggest software engineers to change context node *X8302* of Figure 1a since it is impacted by the change.

We aim to support safety design by developing a framework to propose effective recommendations about the AC that help improve safety analysis. To address this goal, we manifest our intention to use ACs as data.

III. Goals

We intend to pursue the forward-looking idea summarized in Figure 2. The safety design activities *manually* performed by the safety engineers are represented in Figure 2 by the *Safety Argument Design* rectangular box. The changes to claim *C5033* and its corresponding evidence (*E5036*) in Figure 1b described in Section II are examples of these manual safety activites.

The numbered rectangular boxes in Figure 2 report the steps of our proposed approach. These steps are as follows:

Monitoring (1) automatically records the activities performed by safety engineers during the development of the safety argument and returns a sequential log of safety events. For example, changing the claim *C5033* of Figure 1b and its corresponding evidence (*E5036*) would create two new safety events corresponding to node modifications. The safety event logs record the changes applied to the nodes and the date and time at which the changes were made. For example, Listing 1 presents a portion of a log recording the activities the designer performed in creating the safety argument for the ACC, where NC, ND, and NM respectively represent the node creation, deletion, and modification. For the creation of safety nodes, the log reports the time of the change, the ID of the parent node and a textual description. For example, the entry ⟨*2022/06/21-12:00:00, NC, C8300, Root, "When it is considered safe [...]"*⟩ creates a new safety node C8300 with parent Root and text "When it is considered safe [...]" on 2022/06/21-12:00:00. For the deletion of a safety node, the log reports the time of the deletion and the ID of the node. For example, the entry ⟨*2022/06/21-12:07:34, ND, A8303*⟩ removes the node A8303 on 2022/06/21-12:07:34. Finally, for the change of a safety node, the log reports the time, the id of the node its old and new textual description. For example, the entry ⟨*2022/06/21-12:09:08, NM, X8302, "When engaging [...]", "Before eng [...]"*⟩ modifies the node X8302 by changing its text from "When engaging [...]" to "Before eng [...]" on 2022/06/21-12:09:08.

Listing 1: A portion of the safety event log for the ACC.

```
2022/06/21-12:00:00, NC, C8300, Root, "When it is considered safe [...]"
2022/06/21-12:01:31, NC, S8310, C8300, "Argue over the possible [...]"
2022/06/21-12:02:47, NC, A8301, C8300, "If the user engages [...]"
2022/06/21-12:05:19, NC, X8302, C8300, "When engaging, the ACC [...]"
2022/06/21-12:07:06, NC, A8303, C8300, "Assume that the env [...]"
2022/06/21-12:07:34, ND, A8303
2022/06/21-12:09:08, NM, X8302, "When engaging [...]", "Before eng [...]"
```

[...] represents portions of text removed for brevity.

Learning (2): analyzes the safety log and learns relations between safety-related concerns (e.g., safety event relations). For example, by analyzing the safety log, it is possible to detect that the claim *C5033* and the context node *X8302* are frequently changed together. This indicates that it is likely that the two nodes represent related safety concerns.

Recommending (3): provides recommendations about the safety argument to the safety engineer. For example, using the output of the learning component, after the safety engineers change the claim *C5033*, we envision recommending that the context node *X8302* also need to be changed.

IV. Benefits

We envision several benefits from considering ACs as data:

Provide effective recommendations to safety engineers. Analyzing data collected during the AC development can enable learning the relation between different safety concerns (e.g., different nodes of the AC diagram) and produce safety recommendations that can guide the safety analyst. Specifically, related nodes are likely to change subsequently during the AC development. Learning from these relationships between nodes enables effective recommendations to be identified. For example, the recommendation framework can suggest nodes that are most likely to be impacted by a particular change that a developer makes.

Extract metrics and statistics related to AC development. Monitoring and analyzing AC development data can enable engineers to extract metrics and statistics that improve AC development. For example, safety analysts can use these metrics and statistics to identify regions of the AC that are more frequently changed. It is likely that these regions are more safety-critical or contain more uncertain safety aspects, as developers spent more time on their development.

Extrapolate safety development patterns. Analyzing the data collected during the AC development can enable the learning of safety development patterns. These patterns can be categorized by the safety analysts to define common guidelines for AC development. These patterns can standardize AC development and help produce safer systems.

Cross-developer recommendations. Analyzing AC development data can enable cross-developer recommendations.

Specifically, treating ACs as data can enable learning the development styles of experienced safety analysts, and use them to train less experienced personnel.

Standard development compliance. Analyzing AC development data can improve standard compliance. Treating ACs as data can enable monitoring development practices and ensure that compliance expectations are met.

Preliminary evaluation. We assessed the potential benefits of our idea by instrumenting an industrial AC tool (SOCRATES [5]) to collect data and by monitoring the activity of four engineers during the design of an AC for the CERN Large Hadron Collider [19] that took approximately three months. The complete AC produced at the end of this period contained 454 nodes. We analyzed one branch of the argument containing 153 nodes, and collected 784 actions representing node creations, changes, and deletions. We used the CPT+ [20], [21] recommendation framework to learn patterns from the collected data and predict changes in the AC, and compared its accuracy with the accuracy of a random predictor. Since CPT+ requires actions to be split in sequences (i.e., chunks of actions representing a specific development activity), for our preliminary evaluation we split the 784 actions into sequences of five actions. We considered five actions because this decision enabled us to obtain a large number (157) of sequences that can be processed by CPT+. For each sequence, we trained CPT+ by considering the past history (all the past actions before the given sequence). We used the trained model to predict the actions within that sequence and calculated the hit rate (i.e., the percentage of correct predictions within that sequence). Note that the CPT+ algorithm aborts when the trained model is not sufficiently confident in a prediction; this often occurs when new nodes are created, as there are no prior actions related to this node to learn from. We discarded these cases in our evaluation. The random predictor picks an AC node at random from the set of existing nodes at that point in time, and uses it as the prediction. The CPT+ algorithm returned a prediction for 64.03% of the cases. Table I summarizes the results for these cases. The CPT+ predictor was, on average, 11% more effective than the random one (2% vs 13%). The CPT+ predictor could reach a 100% hit rate for some sequences, while the maximum hit rate for the random predictor was 66.67%. The minimum hit rate for both the CPT+ and the random predictor was 0% since, for some sequences, neither of them could correctly predict any change. These preliminary results confirm that using ACs as data can provide recommendations for AC development: the CPT+ was more effective than the random predictor. While our results are not groundbreaking at the moment, performing an extensive evaluation, including considering other policies to identify sequences of actions and other recommendation frameworks, is part of our future work and it will enable us to improve the effectiveness of our solution. Despite these limitations, our preliminary results confirm that there are patterns that can be learned from AC development data, and investigating the usage of AC as data may lead to practical benefits.

TABLE I: Minimum, average, and maximum percentages of correct hits for the CPT+ and random predictors.

Predictor	Minimum	Average	Maximum
CPT+	0.00	13.84	100.00
Random	0.00	2.14	66.67

V. RELATED WORK

Researchers have studied extensively how to support software engineers in the design of safety arguments (e.g., [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [17], [38]). These approaches are supported by existing tools (e.g., [39], [40], [41], [8], [3]). Unlike existing work, our manifesto proposes to treat ACs as data.

Providing recommendations for AC development is significantly different from proposing recommendation frameworks for source code development (e.g., [42], [43]), since the goal of AC is to ensure that there is enough evidence to show that a system is safe. This considerably reduces the amount of available data that can be collected. Therefore, the way to collect and process the data should be significantly different from the solutions developed for source code recommendations.

VI. SUMMARY AND NEXT STEPS

To turn our idea into a practical framework (and a full-length paper), we plan to proceed along the following steps:

Develop an exemplar AC. We are currently developing an industrial AC exemplar by collaborating with CSL. We are recording and monitoring the safety development activities.

Implement the proposed framework. We plan to implement a solution that follows the ideas presented by this manifesto by reusing existing machine learning and monitoring techniques to learn safety event relations and provide recommendations. For example, to learn relations between safety-related concerns from the safety log, we can rely on *sequence patterns mining techniques* (e.g., [44]) that extract patterns from a sequence of data (e.g., web clickstreams, or bio-logical sequences).

Assess and evaluate the proposed approach. We plan to use our exemplar to evaluate the usefulness of our recommendations, and conducting user studies in collaboration with CSL.

To conclude, treating AC development as data may provide significant benefits to software development by enabling safety analysts to produce safer systems and reduce costs. This manifesto issues a broader call to action from other researchers and industries that may help engineers produce safer systems by using ACs as data.

Data Availability. Our data is not released due to an NDA.

ACKNOWLEDGMENT

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) [funding reference numbers RGPIN-2022-04622, DGECR-2022-0040,RGPIN-2015-06366].

We acknowledge the support of CSL: Adam Casey for the ACC example, Mateo Delgado and Rolf Lippelt for the help with our preliminary example, and Simon Diemert for the support in obtaining event logs from the SOCRATES AC tool.

REFERENCES

[1] V. Astvansh, G. P. Ball, and M. Josefy, "The recall decision exposed: Automobile recall timing and process data set," *Manufacturing & Service Operations Management*, 2022.

[2] R. N. Charette, "How software is eating the car," *IEEE Spectrum, June*, 2021.

[3] S. Nair, J. L. De La Vara, M. Sabetzadeh, and L. Briand, "An extended systematic literature review on provision of evidence for safety certification," *Information and Software Technology*, vol. 56, no. 7, pp. 689–717, 2014.

[4] ISO, "ISO26262: Road vehicles – Functional safety," 2011.

[5] "Socrates Assurance Case Editor ," https://safetycasepro.com/welcome, 04 2022 [Online].

[6] "Ansys Safety Analysis Ensure System Safety and Cybersecurity," https://www.ansys.com/products/safety-analysis, 04 2022 [Online].

[7] "Ensure System Safety with GSN," https://astah.net/products/astah-gsn/, 04 2022 [Online].

[8] M. Maksimov, N. L. Fung, S. Kokaly, and M. Chechik, "Two decades of assurance case tools: a survey," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2018, pp. 49–59.

[9] M. Maksimov, S. Kokaly, and M. Chechik, "A survey of tool-supported assurance case assessment techniques," *ACM Computing Surveys*, vol. 52, no. 5, pp. 1–34, 2019.

[10] C. Cârlan, B. Gallina, and L. Soima, "Safety case maintenance: a systematic literature review," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2021, pp. 115–129.

[11] M. Mohamad, J.-P. Steghöfer, and R. Scandariato, "Security assurance cases—state of the art of an emerging approach," *Empirical Software Engineering*, vol. 26, no. 4, pp. 1–43, 2021.

[12] J. L. d. l. Vara, G. Jiménez, R. Mendieta, and E. Parra, "Assessment of the quality of safety cases: A research preview," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2019, pp. 124–131.

[13] V. Sklyar and V. Kharchenko, "Assurance case for safety and security implementation: a survey of applications," *International Journal of Computing*, vol. 19, no. 4, pp. 610–619, 2020.

[14] "manifesto," https://www.merriam-webster.com/dictionary/manifesto, 04 2022 [Online].

[15] "Critical Systems Labs," https://www.criticalsystemslabs.com/, 04 2022 [Online].

[16] GSN Working Group, "GSN Community Standard Version 2," http://www.goalstructuringnotation.info/, York, UK, 2011.

[17] J. Goodenough, C. Weinstock, and A. Klein, "Eliminative argumentation: A basis for arguing confidence in system properties," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2015-TR-005, 2015. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=434805

[18] J. B. Goodenough, C. B. Weinstock, and A. Z. Klein, "Eliminative induction: A basis for arguing system confidence," in *International Conference on Software Engineering*. IEEE, 2013, pp. 1161–1164.

[19] "The Large Hadron Collider," https://home.cern/science/accelerators/large-hadron-collider, 04 2022 [Online].

[20] T. Gueniche, P. Fournier-Viger, and V. S. Tseng, "Compact prediction tree: A lossless model for accurate sequence prediction," in *International Conference on Advanced Data Mining and Applications*. Springer, 2013, pp. 177–188.

[21] T. Gueniche, P. Fournier-Viger, R. Raman, and V. S. Tseng, "Cpt+: Decreasing the time/space complexity of the compact prediction tree," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2015, pp. 625–636.

[22] N. Desai and S. Punnekkat, "Safety-oriented flexible design of autonomous mobile robot systems," in *International Symposium on Systems Engineering*. IEEE, 2019, pp. 1–7.

[23] C. Ponsard, G. Dallons, and P. Massonet, "Goal-oriented co-engineering of security and safety requirements in cyber-physical systems," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2016, pp. 334–345.

[24] G. Forcina, A. Sedaghatbaf, S. Baumgart, A. Jafari, E. Khamespanah, P. Mrvaljevic, and M. Sirjani, "Safe design of flow management systems using rebeca," *Journal of Information Processing*, vol. 28, pp. 588–598, 2020.

[25] S. Björnander, R. Land, P. Graydon, K. Lundqvist, and P. Conmy, "A method to formally evaluate safety case evidences against a system architecture model," in *International Symposium on Software Reliability Engineering Workshops*. IEEE, 2012, pp. 337–342.

[26] R. Calinescu, D. Weyns, S. Gerasimou, M. U. Iftikhar, I. Habli, and T. Kelly, "Engineering trustworthy self-adaptive software with dynamic assurance cases," *IEEE Transactions on Software Engineering*, vol. 44, no. 11, pp. 1039–1069, 2017.

[27] T. P. Kelly and J. A. McDermid, "A systematic approach to safety case maintenance," *Reliability Engineering & System Safety*, vol. 71, no. 3, pp. 271–284, 2001.

[28] L. Murphy, T. Viger, A. D. Sandro, R. Shahin, and M. Chechik, "Validating Safety Arguments with Lean," in *Software Engineering and Formal Methods*. Springer, 2021, pp. 23–43.

[29] T. Viger, L. Murphy, A. Di Sandro, R. Shahin, and M. Chechik, "A Lean Approach to Building Valid Model-Based Safety Arguments," in *International Conference on Model Driven Engineering Languages and Systems*. ACM/IEEE, 2021, pp. 194–204.

[30] T. E. Wang, Z. Daw, P. Nuzzo, and A. Pinto, "Hierarchical contract-based synthesis for assurance cases," in *NASA Formal Methods Symposium*. Springer, 2022, pp. 175–192.

[31] Z. Diskin, T. Maibaum, A. Wassyng, S. Wynn-Williams, and M. Lawford, "Assurance via model transformations and their hierarchical refinement," in *International Conference on Model Driven Engineering Languages and Systems*. ACM/IEEE, 2018, p. 426–436.

[32] R. Shahin, S. Kokaly, and M. Chechik, "Towards certified analysis of software product line safety cases," in *Computer Safety, Reliability, and Security*. Springer, 2021, pp. 130–145.

[33] T. Viger, R. Salay, G. Selim, and M. Chechik, "Just enough formality in assurance argument structures," in *Computer Safety, Reliability, and Security*. Springer, 2020, pp. 34–49.

[34] N. Annable, T. Chiang, M. Lawford, R. F. Paige, and A. Wassyng, "Generating assurance cases using workflow$^+$ models," in *Computer Safety, Reliability, and Security*. Springer, 2022, pp. 97–110.

[35] T. Chowdhury, A. Wassyng, R. F. Paige, and M. Lawford, "Criteria to systematically evaluate (safety) assurance cases," in *International Symposium on Software Reliability Engineering*. IEEE, 2019, pp. 380–390.

[36] N. L. Fung, S. Kokaly, A. Di Sandro, and M. Chechik, "Assurance Case Property Checking with MMINT-A and OCL," in *Recent Trends and Advances in Model Based Systems Engineering*. Springer, 2022, pp. 351–360.

[37] Y. Sirgabsou, C. Baron, L. Pahun, and P. Esteban, "Software fault propagation patterns for model-based safety assessment in autonomous cars," in *European Congress on Embedded Real Time Systems*, 2022.

[38] S. Diemert and J. Joyce, "Eliminative argumentation for arguing system safety-a practitioner's experience," in *International Systems Conference*. IEEE, 2020, pp. 1–7.

[39] N. L. Fung, S. Kokaly, A. D. Sandro, R. Salay, and M. Chechik, "MMINT-A: a tool for automated change impact assessment on assurance cases," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2018, pp. 60–70.

[40] A. Gacek, J. Backes, D. Cofer, K. Slind, and M. Whalen, "Resolute: an assurance case language for architecture models," *ACM SIGAda Ada Letters*, vol. 34, no. 3, pp. 19–28, 2014.

[41] Y. Luo, M. van den Brand, Z. Li, and A. K. Saberi, "A systematic approach and tool support for GSN-based safety case assessment," *Journal of Systems Architecture*, vol. 76, pp. 1–16, 2017.

[42] U. Pakdeetrakulwong, P. Wongthongtham, and W. V. Siricharoen, "Recommendation systems for software engineering: A survey from software development life cycle phase perspective," in *International Conference for Internet Technology and Secured Transactions*. IEEE, 2014, pp. 137–142.

[43] J. Beel, B. Gipp, S. Langer, and C. Breitinger, "Paper recommender systems: a literature survey," *International Journal on Digital Libraries*, vol. 17, no. 4, pp. 305–338, 2016.

[44] E. Salvemini, F. Fumarola, D. Malerba, and J. Han, "Fast sequence mining based on sparse id-lists," in *International Symposium on Methodologies for Intelligent Systems*. Springer, 2011, pp. 316–325.