Towards Automated Design-Time Explainability of Assurance Cases and Trade Spaces

Bryce Bjorkman*, Bryan C. Ward*, Saqib Hasan†
*Vanderbilt University
Email: {bryce.a.bjorkman, bryan.ward}@vanderbilt.edu
†Collins Aerospace
Email: {saqib.hasan}@collins.com

***Abstract*—Formal methods for model-based systems engineering have matured significantly in recent years, with new tools and approaches successfully demonstrated in various safety-critical and real-time domains. However, a key remaining barrier to more widespread adoption of these tools and techniques is explainability. It is impractical to require a user of such tools to understand all of the details of model synthesis, analysis, transformation, and implementation, yet some level of understanding is required to make effective use of these tools. Furthermore, designers must be able to explain their use of the tools to each other and stakeholders to establish confidence in the overall results.**

This paper presents a vision of how different formal methods techniques for model-based systems engineering can be integrated to automate the production of explainable feedback regarding assurance and optimization at design time.

I. Introduction

Model-Based Systems Engineering (MBSE) is "the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases" [1]. Complex systems development benefits from virtual integration [2], in which MBSE techniques are used to integrate potentially disparate models of system components earlier in the development process when the costs to correct design errors tend to be lower. This shift-left approach to validation can be realized by architecture-led development processes, such as the Double V Model shown in Figure 1 wherein validation of the design is performed through virtual integration at the level of requirements, system architecture, software architecture, and software components. Architecture modeling, analysis, and code generation are central to this process as they enable the formal definition, assessment, and refinement of candidate designs.

The main tool needed to support this kind of process is a modeling environment such as OSATE [3], which is centered around the Architecture Analysis and Design Language (AADL) [4] [5], where the formalism is suitable for real-time embedded systems (RTES). Research programs such as HACMS [6] and CASE [7] have demonstrated that real-time systems can be developed to be secure by design using such an environment enriched with model-based verification tools to ensure requirements are met by the design and that the produced implementation conforms to the design.

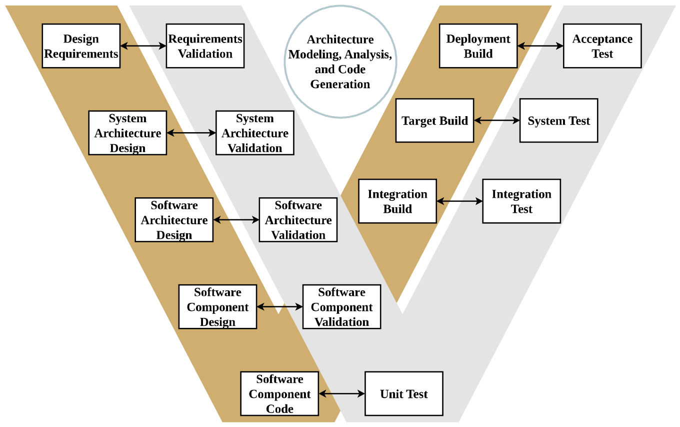
Design Requirements
Requirements Validation
Architecture Modeling, Analysis, and Code Generation
Deployment Build
Acceptance Test
System Architecture Design
System Architecture Validation
Target Build
System Test
Integration Build
Integration Test
Software Architecture Design
Software Architecture Validation
Software Component Design
Software Component Validation
Software Component Code
Unit Test

Fig. 1. Double V Model of Development and Assurance

We need assurance of both security and timeliness for safety-critical RTES, but design mechanisms for achieving either of these goals oftentimes negatively impact the pursuit of the other. Oftentimes, system designers must balance additional competing design objectives such as optimizing for size, weight, and power (SWaP), as well as cost. Furthermore, as RTES solutions grow more complex and developers increasingly rely on artificial-intelligence copilots, the justifications that a given design meets its requirements grow more complex and rely on expert knowledge that is potentially opaque to developers and stakeholders alike.

This presents issues in terms of both explainability for why a given design does or does not meet certain requirements and optimization of other properties such as SWaP. Though modern architecture-led processes using formal methods enable secure-by-design development of safety-critical RTES, we believe that it is important to refine these ideas and define an explainability-oriented development process in order to better solve the problem of providing assurance for safety-critical systems. In this paper we propose an explainability-oriented development process based on existing architecture-led processes.

II. Security Explainability Challenges

Measuring security is notoriously difficult [8]. For instance, empirical measurements of the resilience of a given cryptographic scheme to a given attack cannot prove that the cryptographic scheme is truly secure against the attack, and cannot prove that some other attack against which the scheme is not resilient does not exist. For those reasons, the cryptography community tends to rely on reductionist proofs in which the security of a scheme is tied to the security of its underlying atomic primitives, which are widely assumed to be secure due to the mathematical difficulty of breaking them. This reliance on deductive reasoning is shared by the formal-methods community, wherein techniques such as model checking are used to prove properties of a system model, as well as by the real-time systems community, wherein mathematical analyses are used to prove timing bounds for workloads under given task models and scheduling disciplines. Such deductive reasoning about a model from assumed properties has the advantage of certainty, but requires being explicit about the assumptions on which the reasoning are conditioned because, while the results certainly hold for the model, they can only hold for the real system if it meets those assumptions. While inductive reasoning using empirical measurements cannot prove the security of a system in general, it remains our only means of assuring that a system conforms to the specific model and assumptions under which formal analysis using deductive reasoning can prove security. A specific security property can be proven for a given system model and threat model at design time, but we must have assurance that the implemented system conforms to the design in order to gain confidence that it meets the design assumptions. We cannot prove everything about a system, but we want explainability for what we can.

So, for an assurance case to provide explainability of security, it must provide the threat models that are considered, the proof of system model security against those threat models, the assumptions that the implemented system must satisfy for the model-based argument to transfer, and results of testing to show that the implemented system meets those assumptions.

III. Related Work

Research programs such as HACMS [6] and CASE [7] have made use of the assurance-case language Resolute [9], the compositional reasoning tool AGREE [10], the information flow and error-propagation-analysis framework Awas [11], the requirements discovery tools GearCASE [12] and DCRYPPS [13], the code generation toolset HAMR [14], and the formally verified microkernel seL4 [15]. Using these tools together allows for automatically maintaining a formal argument of implementation faithfulness to design from the highest levels of abstraction down to low level operating system code. We see this through-line as a boon to explainability since assurance arguments presented at higher levels of abstraction do not necessarily need to be understood at lower levels of abstraction for a stakeholder to accept them.

Assuring that a design meets requirements in general has been studied with formalized approaches such as rich traceability [16], where requirement satisfaction arguments are structured as combinations of statements regarding the design that can be justified using supporting evidence. In terms of assurance for certain security properties at the architectural level of abstraction, recent works [17] [18] have explored the application of zero trust design and assurance patterns for cyber-physical systems. The zero trust architecture paradigm [19] achieves security by following several tenets aimed at eliminating perimeter-based trust zones and relying instead on continuous authentication and verification. Just as reusable design patterns [20] can be instantiated to form parts of a larger system architecture, assurance patterns [21] can be reused to form parts of a larger assurance case, providing claims and arguments about the safety [22] and security [23] properties that a corresponding design pattern provides. For instance, a designer may choose to use a zero trust design and assurance pattern such as a policy enforcement point to mitigate the threat of a compromised component making unauthenticated access to the interface of another component. The design pattern provides information necessary to transform the system model, while the assurance pattern provides information necessary to support the claim that the threat is mitigated in the overall system design. Relying on such pattern-based security mechanisms helps keep design security verifiable and explainable.

The Systems-Theoretic Accident Model and Processes (STAMP) [24] [25] is an accident causality model that expands on traditional chain-of-component-failure models by considering hazards introduced by emergent properties of a composed system. The System-Theoretic Process Analysis (STPA) [26] [25] is then a hazard analysis process that assumes the STAMP model and can uncover hazards dependent on emergent properties that traditional approaches such as Fault Tree Analysis cannot. In STPA, the focus of analysis is on high-level control structures and the interfaces between them. The process consists of identifying what losses must be prevented, modeling the high-level control structures present in the system, identifying what unsafe control actions could cause losses, and then identifying what scenarios could lead to those unsafe control actions. The adaptation of STPA for security (STPA-sec) [27] meshes well with our focus on architectural models and information flow because we can consider what communications between components may result in change in security-related state. We see the identified loss scenarios produced by this process as crucial artifacts for explainable security.

Combining the principles of STPA and virtual integration, Architecture-Led Safety Analysis (ALSA) [28] and Systematic Analysis of Faults and Errors (SAFE) [29] [30] are approaches related to assurance that identify system assumptions and trace hazards to architectural elements at different levels of the design hierarchy. Building on the ALSA approach by automating assurance case verification, Automated Assurance of Security Policy Enforcement (AASPE) [31] references both the Common Weakness Enumeration (CWE) database [32] and the Multiple Independent Layers of Security (MILS) architecture [33] when performing attack impact analysis and attack tree analysis to determine losses and paths to achieve those losses, as well as whether security policies block attack

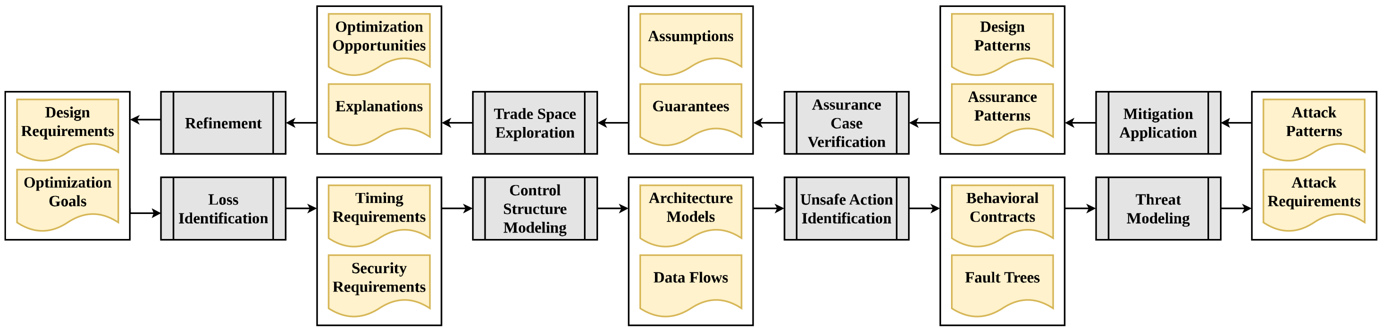

Fig. 2. Explainability-Oriented Development Process

paths and whether the model faithfully enacts those security policies. We see the referencing of a corpora of threat models in this approach as a key part of explainable security.

In terms of tradeoff optimization, Guided Architecture Trade Space Exploration (GATSE) [34] is a recent work that makes use of the ARL Trade Space Visualizer (ATSV) [35] to guide a user through design-space exploration by allowing them to define decision variables and then presenting a set of design alternatives alongside their impact on the decision variables. Importantly, it supports constraining decision variables to preclude infeasible design alternatives. We see this as an extensible way for designers to find explanations of the ways in which their design constraints admit opportunities for optimization.

With regard to security-oriented optimization, QUASAR [36] and QUASAR^{RT} [37] developed strategies for finding optimal design points when trading performance for security. They developed attack capability graphs for memory corruption vulnerabilities and used them as a basis for a security metric that could be weighted against traditional real-time performance metrics. We see this new form of security metric as a key enabler of explaining design tradeoffs, since it is often desirable to add security mechanisms up to the point of violating performance constraints, but deciding which set of security mechanisms to add is non-trivial without additional information.

Overall, there is clear groundwork in the literature for an explainability-oriented development process that supports both assurance and optimization.

IV. Proposed Solution

We propose a formal-methods approach to secure real-time system design that is conducive to both explainable assurance and explainable optimization. We feel that it is achievable to have a highly automated, iterative, model-based design process based on the groundwork laid by related works, where our analyses provide feedback to the designer in the form of both assurance-case verification results and optimization metrics.

Figure 2 depicts our idea of what this iterative process could look like. The process consists of requirements-driven generation based on STPA and analysis-driven refinement based on feedback. First, a designer begins with their top-level design requirements and optimization goals. Then, a designer must identify losses that should be prevented. These losses consist of timing requirements and security requirements that must be satisfied for the system to be safe and secure. Then, the designer undertakes top-down modeling of their system architecture, focusing on the control structure, which is the set of components that communicate information that controls the state and behavior of other components.

An important aspect of this step is the annotation of component processes, flows, data, and ports with labels connected to the requirements. For timing requirements, this may consist of annotating processes with periods and deadlines, and annotating flows with maximum tolerable latencies. For security requirements, this may consist of annotating data as critical protected information or unvalidated input, and annotating ports with privileges. The architecture models and data flows produced by this step are then used in identifying unsafe actions that could occur based on communication between components. This produces behavioral contracts specifying what well-behaved components are expected to do, and fault trees based on data dependencies that specify what adverse outcomes can occur if components fail to behave.

Having specified what is needed to meet design requirements and what must be protected, this takes us to the step of threat modeling, wherein tools help the designer identify scenarios in which attacks could compromise components or otherwise cause contract violations. This step produces a set of applicable attack patterns and capabilities that are required for those attacks to be feasible. In order to stop these scenarios from occurring, the designer moves on to applying mitigations, transforming the system model using the appropriate zero trust design patterns and updating the system assurance case with their corresponding assurance patterns.

Now, at this point in the development cycle, the assurance case that has been built thus far can be verified through automatic argumentation, such as in the Symbolic Assurance Refinement framework [38]. The verified properties then act as constraints on subsequent trade space exploration, which produces optimization opportunities and explanations of how much freedom remains in the design space. Lastly, the designer refines their design requirements and optimization goals, taking the feedback from analyses performed earlier in the development process into account.

We believe our approach could form the basis for continuous assurance in MBSE similar to CI/CD in modern software engineering. In both CI/CD and our approach, though the process is depicted as a continuous cycle, failing to pass tests or find feasible paths forward may result in backtracking to a previous design point.

V. Optimization

We see our proposed solution as an enabler of more efficient design-space exploration, and one specific way that it could support this is through automated analysis of design alternatives.

Consider, for instance, a set of security mechanisms available in an AADL library such as the recently explored zero trust design patterns of [18]. A designer may wish to transform their model using these patterns to increase security, but adding certain components from the library may only have marginal security benefits but with significant performance impacts. Perhaps the designer wants to improve the security of a data flow between two systems, in which case their most secure option may be to employ encryption, signing, and attestation. If the flow is of high criticality and the targeted platforms can support all three mechanisms without sacrificing performance requirements such as workload schedulability and communication latency, then the designer may well choose all three. However, if using all three would make performance requirements unsatisfiable, then the designer should be provided with explanations of how each candidate design pattern impacts performance and security in order to make their decision. This kind of decision problem is common in the design of mission critical systems, where a mission's overall success is highly dependent on meeting the objectives of the mission within the required timing constraints.

In another example, consider the use of a separation kernel such as seL4. Arguably, the most secure use of the separation kernel would be to put every thread of a system into its own protection domain. However, for every flow of information and piece of shared state between threads, the imposed separation would incur overheads due to inter-process communication and synchronization. Again, if total separation of all threads is feasible under the design constraints then that may be the designer's choice, but if not then the designer would need some additional information on the relative security value of each separation option.

These kinds of tradeoffs can be difficult for a systems engineer to evaluate on a case-by-case basis, but the evaluation becomes even more fraught when considering the interplay between disparate security mechanisms. What if the choice is between adding attestation here or separating these threads there? Changes in functional requirements can invalidate previous choices of security mechanisms as well. What if the system now has to perform some additional signal processing or real-time inference? Furthermore, these kinds of tradeoffs can present themselves across different nodes in a distributed system. What if we want to use signing for authentication between two nodes, but the communication latency would then violate performance requirements? Could some components on either node be placed in the same protection domain to take advantage of shared memory? Should symmetric encryption keys be distributed to the nodes before deployment to remove latency due to key establishment? Is there a lighter-weight authentication protocol that could be used instead of the currently selected candidate? With better explainability of the security and performance impacts of design choices presented in assurance cases via automated analyses, we believe designers will also be better equipped to optimize their designs earlier in the development process without sacrificing flexibility to respond to changing requirements.

VI. Conclusion

This paper presented a vision of how different formal methods techniques for model-based systems engineering can be integrated to automate the production of explainable feedback regarding assurance and optimization at design time. To the best of our knowledge, there is no available tool that fully supports the approach outlined in this paper, where, on each iteration of design, the architecture models are analyzed to verify that the design meets both timing and security requirements, to find fault conditions under which the requirements are not met, to find attack patterns that could cause those fault conditions, then transformed using design patterns to mitigate those threats, updating and verifying the system assurance case, using the assumed constraints and guarantees of the assurance case to drive trade space exploration, and providing explanations of how the current design choices affect optimization opportunities to guide refinement on subsequent iterations.

While most parts of the approach are supported by some tools within the OSATE ecosystem, their integration and automation remain as challenges. Some parts of the approach require further investigation, such as modeling the performance versus security tradeoffs when applying zero trust design patterns. In our future work, we aim to refine such tools and to develop a framework implementing our approach that can handle the automation of the entire tool set to seamlessly showcase the design trade-offs given the targeted safety and security requirements.

References

[1] T. INCOSE, "Systems engineering vision 2020," *INCOSE, San Diego, CA, accessed Jan*, vol. 26, no. 2019, p. 2, 2007.

[2] D. Redman, D. Ward, J. Chilenski, and G. Pollari, "Virtual integration for improved system design," in *First Analytic Virtual Integration of Cyber-Physical Systems Workshop*. Citeseer, 2010, pp. 57–64.

[3] Carnegie Mellon University, Software Engineering Institute, "Osate: Open source aadl tool environment," 2025. [Online]. Available: https://osate.org

[4] P. H. Feiler, B. Lewis, S. Vestal, and E. Colbert, "An overview of the sae architecture analysis & design language (aadl) standard: A basis for model-based architecture-driven embedded systems engineering," in *Architecture Description Languages*, P. Dissaux, M. Filali-Amine, P. Michel, and F. Vernadat, Eds. Boston, MA: Springer US, 2005, pp. 3–15.

[5] SAE International, "Architecture analysis & design language (aadl)," Standard AS5506D, Warrendale, PA, USA, 2022. [Online]. Available: https://www.sae.org/standards/content/as5506d/

[6] D. Cofer, A. Gacek, J. Backes, M. W. Whalen, L. Pike, A. Foltzer, M. Podhradsky, G. Klein, I. Kuz, J. Andronick, G. Heiser, and D. Stuart, "A formal approach to constructing secure air vehicle software," *Computer*, vol. 51, no. 11, pp. 14–23, 2018.

[7] D. Cofer, I. Amundson, J. Babar, D. Hardin, K. Slind, P. Alexander, J. Hatcliff, Robby, G. Klein, C. Lewis, E. Mercer, and J. Shackleton, "Cyberassured systems engineering at scale," *IEEE Security & Privacy*, vol. 20, no. 3, pp. 52–64, 2022.

[8] C. Herley and P. Van Oorschot, "Sok: Science, security and the elusive goal of security as a scientific pursuit," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 99–120.

[9] I. Amundson and D. Cofer, "Resolute assurance arguments for cyber assured systems engineering," in *Proceedings of the Workshop on Design Automation for CPS and IoT*, ser. Destion '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 7–12. [Online]. Available: https://doi.org/10.1145/3445034.3460507

[10] D. Cofer, A. Gacek, S. Miller, M. W. Whalen, B. LaValley, and L. Sha, "Compositional verification of architectural models," in *NASA Formal Methods*, A. E. Goodloe and S. Person, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 126–140.

[11] H. Thiagarajan, J. Hatcliff, and Robby, "Awas: Aadl information flow and error propagation analysis framework," *Innovations in Systems and Software Engineering*, vol. 18, no. 4, pp. 485–504, Dec 2022. [Online]. Available: https://doi.org/10.1007/s11334-021-00410-w

[12] T. Patten, D. Mitchell, and C. Call, "Cyber attack grammars for risk/cost analysis," pp. 597–605,XVI, 03 2020, copyright - Copyright Academic Conferences International Limited Mar 2020; Last updated - 2024-08-27. [Online]. Available: https://www.proquest.com/conference-papers-proceedings/cyber-attack-grammars-risk-cost-analysis/docview/2455899797/se-2

[13] R. Laddaga, P. Robertson, H. Shrobe, D. Cerys, P. Manghwani, and P. Meijer, "Deriving cyber-security requirements for cyber physical systems," 2019. [Online]. Available: https://arxiv.org/abs/1901.01867

[14] J. Hatcliff, J. Belt, Robby, and T. Carpenter, "Hamr: An aadl multi-platform code generation toolset," in *Leveraging Applications of Formal Methods, Verification and Validation*, T. Margaria and B. Steffen, Eds. Cham: Springer International Publishing, 2021, pp. 274–295.

[15] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "sel4: formal verification of an os kernel," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, ser. SOSP '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 207–220. [Online]. Available: https://doi.org/10.1145/1629575.1629596

[16] J. Hammond, R. Rawlings, and A. Hall, "Will it work? [requirements engineering]," in *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, 2001, pp. 102–109.

[17] S. Hasan, I. Amundson, and D. Hardin, "Zero trust architecture patterns for cyber-physical systems," *SAE International Journal of Advances and Current Practices in Mobility*, vol. 5, no. 2023-01-1001, pp. 1919–1931, 2023.

[18] ——, "Zero-trust design and assurance patterns for cyber–physical systems," *Journal of Systems Architecture*, vol. 155, p. 103261, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138376212400198X

[19] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero trust architecture," 2020-08-10 04:08:00 2020. [Online]. Available: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=930420

[20] K. Beck and R. Johnson, "Patterns generate architectures," in *Object-Oriented Programming*, M. Tokoro and R. Pareschi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 139–149.

[21] I. Šljivo, G. J. Uriagereka, S. Puri, and B. Gallina, "Guiding assurance of architectural design patterns for critical applications," *Journal of Systems Architecture*, vol. 110, p. 101765, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138376212030059X

[22] E. Denney and G. Pai, "A formal basis for safety case patterns," in *Computer Safety, Reliability, and Security*, F. Bitsch, J. Guiochet, and M. Kaâniche, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 21–32.

[23] C. B. Weinstock, H. F. Lipson, and J. Goodenough, "Arguing security-creating security assurance cases," Carnegie Mellon University, Software Engineering Institute, Tech. Rep., 2007. [Online]. Available: https://apps.dtic.mil/sti/html/trecms/AD1171194/

[24] N. Leveson, "A new accident model for engineering safer systems," *Safety Science*, vol. 42, no. 4, pp. 237–270, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092575350300047X

[25] N. G. Leveson, *Engineering a Safer World: Systems Thinking Applied to Safety*. The MIT Press, 01 2012. [Online]. Available: https://doi.org/10.7551/mitpress/8179.001.0001

[26] N. G. Leveson and J. P. Thomas, "Stpa handbook," Massachusetts Institute of Technology, Partnership for Systems Approaches to Safety and Security (PSAS), Tech. Rep., 2018.

[27] W. Young and N. Leveson, "Systems thinking for safety and security," in *Proceedings of the 29th Annual Computer Security Applications Conference*, ser. ACSAC '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 1–8. [Online]. Available: https://doi.org/10.1145/2523649.2530277

[28] P. Feiler, J. Delange, D. Gluch, and J. D. McGregor, "Architecture-Led Safety Process," Carnegie Mellon University, Tech. Rep., 6 2018. [Online]. Available: https://kilthub.cmu.edu/articles/report/Architecture-Led_Safety_Process/6572033

[29] S. Procter, "A development and assurance process for medical application platform apps," Ph.D. dissertation, Kansas State University, 2016. [Online]. Available: https://www.proquest.com/dissertations-theses/development-assurance-process-medical-application/docview/1842428374/se-2

[30] S. Procter, E. Y. Vasserman, and J. Hatcliff, "Safe and secure: Deeply integrating security in a new hazard analysis," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ser. ARES '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3098954.3105823

[31] P. Feiler and S. Proctor, "Automated assurance of security policy enforcement," 2017. [Online]. Available: https://www.sei.cmu.edu/library/automated-assurance-of-security-policy-enforcement-2017/

[32] T. M. Corporation, "Cwe," 2025. [Online]. Available: https://cwe.mitre.org/

[33] J. Alves-Foss, C. Taylor, and P. Oman, "A multi-layered approach to security in high assurance systems," in *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*, 2004, pp. 10 pp.–.

[34] S. Procter and L. Wrage, "Guided architecture trade space exploration: Fusing model based engineering & design by shopping," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2019, pp. 117–127.

[35] G. Stump, M. Yukish, J. Martin, and T. Simpson, "The arl trade space visualizer: An engineering decision-making tool," in *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2004. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2004-4568

[36] R. Skowyra, S. R. Gomez, D. Bigelow, J. Landry, and H. Okhravi, "Quasar: Quantitative attack space analysis and reasoning," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, ser. ACSAC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 68–78. [Online]. Available: https://doi-org.proxy.library.vanderbilt.edu/10.1145/3134600.3134633

[37] C. Lemieux-Mack, K. Leach, N. Zhang, S. Baruah, and B. C. Ward, "Optimizing runtime security in real-time embedded systems," in *Proc. of Workshop on Optimization for Embedded and Real-time Systems (OPERA)*, 2024.

[38] D. de Niz, B. Andersson, M. H. Klein, J. Lehoczky, H. Kim, G. Romanski, J. Preston, F. Fazi, D. Shapiro, D. C. Schmidt, R. Koontz, and S. Procter, "Flight incident analysis through symbolic argumentation," in *2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC)*, 2024, pp. 1–10.