Qualification of Complex Pre-existing Software for Safety-critical Automotive Systems

Jan Toennemann

Qualification of Complex Pre-existing Software for Safety-critical Automotive Systems

Jan Toennemann
Department of Embedded Systems
Vector Informatik GmbH
Stuttgart, Germany
jan.toennemann@vector.com

***Abstract*—Development of safety-related systems and of the software therein is required to adhere to specific process requirements. Appropriate re-use of existing software, either of legacy in-house projects or open-source solutions, is a reasonable desire among OEMs and suppliers alike. Unfortunately the hurdles are often so paramount that an entire re-engineering of the pre-existing software element results in less overall effort.**

We are aggregating information regarding the largest challenges in this domain to combine them with novel approaches to enable the overall use of pre-existing software for safety-critical automotive systems. This ties in with the efforts made in scope of the ISO/AWI PAS 8926 "Qualification of pre-existing software products for safety-related applications" to cause a shift in the industry and dissipate barriers that are in the way of progress, especially with regard to cross-company collaboration on shared libraries. The intention is to provide at least the same amount of safety evidence as is achieved using conventional development approaches by leveraging the strengths of distributed development processes and modern tooling designed for use in collaborative environments utilizing version control.

***Index Terms*—functional, safety, automotive, legacy, open-source, pre-existing, software, architecture, qualification**

I. Introduction

Software re-use offers significant advantages in meeting the domain-specific requirements of the automotive industry. It enables accelerated development, cost savings, improved efficiency, and enhanced quality. Legacy software, with its established track record, brings reliability and compatibility, while open-source software benefits from community collaboration, customizability, and flexibility. Leveraging both legacy and open-source software components allows automotive manufacturers and suppliers to achieve efficiency, reduce development costs, and accelerate innovation within this industry. The practice of software re-use plays a pivotal role in addressing the unique challenges and demands of the automotive domain, providing a pathway to meet customer expectations and achieve business objectives.

Software for use in safety-critical systems requires tailoring to meet specific requirements and unless the software was explicitly developed for these use cases, additional work needs to be put into achieving compliance. Manufacturers can modify and adapt existing components to suit their unique needs, ensuring the seamless integration of safety functions within the overall vehicle architecture. While this requires use case-specific adjustments for application-level functions, libraries and - in more general terms - basic software can be qualified to meet the requirements of a large scope of potential use cases.

Open-source software offers a cost-effective alternative to proprietary solutions, reducing the burden of licensing fees and enabling organizations to allocate resources towards other critical areas. By leveraging the collective knowledge and expertise of developers worldwide, automotive manufacturers and suppliers can tap into a vast pool of talent, accelerating innovation and facilitating the exchange of best practices. The open-source model encourages community-driven development, ensuring that cutting-edge advancements and bug fixes are shared and improved upon rapidly. In the highly competitive automotive industry, where profit margins are often razor-thin, the cost savings achieved through the utilization of open-source software can provide a significant advantage. This practice is already common in areas like infotainment, but work needs to be put in to ensure that this software can be used in more critical systems without compromising on safety, security, or quality.

The international standard ISO 26262 [1] outlines the functional safety requirements for automotive systems. It provides a framework for developing software in safety-critical automotive systems, with the goal of ensuring the safety of passengers, pedestrians, and the overall environment. However, achieving ISO 26262 compliance poses several challenges for software engineers. With the first version of the standard published back in 2011, there already are several established routes in effect to achieve compliance, focusing on both process as well as technical measures to meet the standard's requirements.

With regard to the specific topic of software re-use, work is currently ongoing to finalize and publish the ISO/AWI PAS 8926 [2], intending to provide a framework for the qualification of pre-existing software elements for safety-related applications. The focus lies on providing an extensive guide to integrate existing software elements into the systems architectures for safety-critical vehicle projects. In its current draft state, process steps include an impact analysis with respect to the intended use case, a suitability evaluation of the pre-existing software element including a classification based on structured properties as well as a determination of required verification activities to achieve ISO 26262 compliance.

II. Successful Implementations of Open-Source Software in the Automotive Industry

The automotive industry has already witnessed successful deployments of open-source software in various domains, demonstrating its viability and potential for safety-critical systems. Open-source platforms such as Android Automotive and Linux-based systems have gained traction in the development of in-vehicle infotainment systems [3]–[6]. These platforms offer customizable user interfaces, support for smartphone integration, and the ability to leverage a vast ecosystem of applications and services, enhancing the overall development experience.

Open-source software frameworks like ROS (Robot Operating System) have emerged as a cornerstone for development of automotive vehicles [7]. ROS provides a robust foundation for sensor integration, perception, motion planning, and control, enabling collaborative development and fostering interoperability among different hardware and software components. Existing implementations suited for the use in safety-critical systems are still based on proprietary forks of the open-source repositories, investing substantial amounts of work into achieving standards compliance that is not provided back to the upstream repository. While contracted support is typically required for software used in safety-related systems and would need to come from companies with a proven track record, the qualification artefacts could be made part of the public repositories; in addition, the need for a fork results in a divergence between the safety-compliant version of the framework and the open-source implementation, requiring a migration that is not as seamless as developers typically intend it to be.

III. Software Re-Use

Software re-use plays a crucial role in modern software development, offering numerous benefits such as reduced development time, cost savings, improved quality, and increased productivity. However, the process of reusing software, particularly when considering libraries developed for different use cases, presents several challenges that need to be addressed.

When reusing software libraries developed for another use case, ensuring the alignment of requirements can be a significant undertaking. The original library may have been designed to meet specific functional and non-functional requirements that differ from those of the new use case. As a result, additional engineering effort is required to analyze, modify, and adapt the library to satisfy the specific requirements of the new application.

Integrating a library developed for a different use case into a new software system requires careful consideration of the overall system design. The reusability of the library depends on its compatibility with the architecture, interfaces, and data models of the new system. Incompatibilities may arise, necessitating architectural adjustments or the development of adapters and middleware to bridge the gaps between the library and the new system. This integration effort can add complexity and overhead to the reusability process.

Ensuring the reliability and correctness of reused software libraries is crucial, as they directly impact the overall quality, safety, and security of the new system. Reused libraries may require extensive testing and validation to verify their functionality, performance, and compatibility with the new use case. Testing efforts may include specifying additional test cases, conducting integration testing, and performing system-level verification to ensure that the library functions as intended within the new context.

Proprietary legacy software may lack active development support and sufficient documentation. This scarcity of available resources can hinder the reusability process, as developers may face difficulties in understanding the inner workings of the software and its potential integration challenges. This limitation may result in increased engineering effort to analyze and adapt the software for the new use case.

IV. Additional Challenges in Safety-related Systems Development

Safety-related systems in automobiles must adhere to rigorous certification processes and comply with industry standards. Incorporating open-source or legacy software software introduces additional complexity in demonstrating compliance, as it requires comprehensive verification and validation procedures to ensure the reliability and safety of the integrated software components.

Developing ISO 26262-compliant software necessitates a comprehensive understanding of the system's safety requirements and potential hazards. Conducting a thorough risk assessment and defining safety goals are crucial steps. Challenges arise in accurately identifying and evaluating potential risks, as well as setting appropriate safety goals that align with the system's intended functions and safety requirements.

The functional safety concept forms the basis for achieving ISO 26262 compliance. Developing a robust functional safety concept involves determining the necessary safety functions, defining safety mechanisms, and allocating safety requirements to software components. Challenges emerge in ensuring that the functional safety concept adequately addresses all potential hazards and satisfies the standard's requirements.

Capturing, analyzing, and managing safety-related software requirements in a clear and unambiguous manner can be challenging. It requires aligning functional requirements as well as safety requirements, maintaining traceability, and ensuring that respective requirements are testable. Designing the software architecture to implement safety functions and meet safety requirements poses challenges in terms of selecting appropriate design patterns, ensuring fault tolerance and redundancy, and managing the complexity of the system. Developing software that adheres to ISO 26262 standards involves activities such as ensuring adherence to coding guidelines, managing code complexity, as well as incorporating fault-detection mechanisms. Validating the safety-related software components through testing, verification, and validation is a complex and time-consuming process. Challenges include specifying and developing comprehensive test cases, performing fault injection

testing, and ensuring the proper integration and compatibility of software components.

Several established routes can be undertaken to achieve ISO 26262 compliance. Automotive SPICE (Software Process Improvement and Capability Determination) is a framework for assessing and improving software development processes in the automotive industry. Aligning with Automotive SPICE guidelines can provide a structured baseline to meet ISO 26262 compliance. Utilizing established safety standards and best practices, such as MISRA C for coding guidelines or AUTOSAR for the automotive software architecture, can help in aligning the development process with ISO 26262 requirements to a certain extent. Employing development tools that specifically support ISO 26262 compliance, such as safety analysis tools, static analysis tools, and automated testing frameworks, can aid in meeting the standard's requirements. Collaborating with third-party assessors who specialize in ISO 26262 compliance can provide expert guidance, assist in ensuring adherence to the standard, and in conducting audits.

There still is no systematic approach to resolve the delta between the outputs of existing, non-compliant development process and the requirements imposed upon software for safety-critical automotive systems. While qualification has been successfully performed for several open-source libraries, larger and more complex projects typically present hurdles that have not yet been overcome. Efforts to engineer an ASIL-compliant Linux distributions have been ongoing for around a decade, yet there is still no public release of such an operating system available. Current ASIL-compliant POSIX implementations are based on closed-source implementations, e.g. BlackBerry QNX, sysgo PikeOS, Wind River vxWorks, or GreenHills INTEGRITY.

Challenges in qualification are not only related to processual difficulties, e.g. providing evidence of standards-compliant development, proper issue reporting, or adherence to open-source licenses, but also on architectural level. With the software primarily being developed for use outside of safety-critical systems, requirements on architectural complexity, specification, design, and test coverage are typically not complied with.

V. Research Overview

The presented problem is neither new nor limited to the automotive domain. Efforts have been performed in the past already attempting to certify legacy automotive software as well as open-source software for use in automotive systems [8], [9]. A systematic mapping of open-source software in safety-critical systems had already been performed after the initial release of the ISO 26262 [10]. Research into these domains is not scoped to open-source software exclusively, but also takes into account pre-existing, uncertified software in general [11]. Similar challenges exist in related domains as well, e.g. aerospace (cp. e.g. [12], [13]), railway (cp. e.g. [14], [15]), the process industry (cp. e.g. [16]), as well as the medical domain (cp. e.g. [17]–[19]).

With regard to the challenges present in the engineering of safety-related systems within the automotive domain, existing efforts have done well to describe the general process requirements for the engineering [20], [21] as well as the requirements on building a comprehensive safety case, both within ISO 26262 definitions [22] as well as in pre-ISO 26262 development [23].

There has been research on refactoring of existing software in parallel to development to achieve compliance [24], on incremental verification of systems [25] and methods to re-use safety case argumentations [26]. Especially with regard to software initially intended for other use cases but also for general process improvement, automatic generation of some of the required artefacts has been examined for automotive [27], rail [28] and adjacent domains. Bridging the gap between safety and security, cross-domain approaches like coverage-guided fuzz testing have also gained traction for the semi-automated verification of safety-related systems [29].

Gaps exist in publicly available research especially with regard to structured approaches on qualification of complex pre-existing software. While the ISO/AWI PAS 8926 intends to close that gap for systems engineering, we are not aware of work focusing on the domain of providing re-usable safety cases for software initially - and potentially in parallel - developed for different use cases. One key point that we have addressed as currently unresolved is how to prevent divergence from repositories that are publicly maintained and where implementation and documentation changes are in control of a larger group representing different interests. It appears inevitable to maintain a dedicated fork for safety-related use, but with it being publicly available and potentially only a short time period behind the release of the main repository, there is potential to unify the working modes that previously appeared to be contradictory.

With open-source software relying on the sharing of code and community contributions, concerns are raised regarding intellectual property rights and security vulnerabilities [30]. Suppliers and manufacturers must establish robust governance frameworks, implement rigorous code reviews, and have mechanisms in place to identify and mitigate potential security risks associated with any used open-source components. Maintaining forked branches helps with overcoming these potential issues to a large extent.

VI. Conclusion

Open-source software holds immense promise for the advancement of safety-critical automotive systems. With collaborative development, cost-effectiveness, and customization capabilities, it has the potential to drive innovation and accelerate safety-related development in the automotive industry. However, challenges related to certification, compliance, intellectual property, and security must be effectively addressed to fully leverage the benefits of open-source software in safety-related systems.

Software re-use offers numerous advantages in terms of efficiency and productivity. However, reusing software libraries developed for different use cases poses challenges in terms of requirements alignment, design integration, and testing efforts.

Open-source software provides advantages such as community support, transparency, and flexibility, which can mitigate some of these challenges. In contrast, proprietary legacy software may present additional hurdles related to incomplete artefacts, licensing, and limited support. When considering software re-use, engineers must carefully evaluate the characteristics and constraints of both open-source and proprietary legacy software to determine the most suitable approach based on the project-specific requirements and circumstances.

Developing ISO 26262-compliant software for safety-critical automotive systems presents significant challenges. These encompass proper risk assessment, derivation of safety goals, a solid functional safety concept, and the complexities associated with requirements engineering, architectural design, coding, and verification/validation activities. Meeting these challenges is essential to ensure the safety, security, and reliability of software in safety-critical automotive systems.

We are certain that these hurdles can be overcome and are actively working on strategies towards a consistent and unified engineering and development approach that enables re-use of both pre-existing as well as open-source software within safety-related systems in the automotive domain.

References

[1] ISO, "ISO 26262: Road vehicles – Functional safety," 2018.

[2] ——, "ISO/AWI PAS 8926: Road vehicles — Functional safety — Qualification of pre-existing software products for safety-related applications (draft version)," 2023.

[3] Aptiv, "Android Automotive Transforms Vehicle Infotainment," https://www.aptiv.com/docs/default-source/white-papers/2020-aptiv-whitepaper-native-google-android-v.pdf?sfvrsn=833c43d_15, Tech. Rep., 2020.

[4] P. Sivakumar, R. S. Sandhya Devi, A. Neeraja Lakshmi, B. VinothKumar, and B. Vinod, "Automotive Grade Linux Software Architecture for Automotive Infotainment System," in *2020 International Conference on Inventive Computation Technologies (ICICT)*, 2020, pp. 391–395.

[5] S. Usorac and B. Pavkovic, "Linux container solution for running Android applications on an automotive platform," in *2021 Zooming Innovation in Consumer Technologies Conference (ZINC)*, 2021, pp. 209–213.

[6] S. Usorac, D. Bogdanovic, D. Peric, and Z. Lukac, "Adding Android capabilities in automotive Linux infotainment: available virtualization technologies," in *2021 29th Telecommunications Forum (TELFOR)*, 2021, pp. 1–4.

[7] N. Puthoff, "How to Achieve Production-Grade Deployment with ROS 2 and RTI Connext," https://content.rti.com/whitepaper-how-to-achieve-production-grade-deployment-with-ros-2-and-rti-connext, RTI, Tech. Rep., 2022.

[8] S. Kochanthara, Y. Dajsuren, L. Cleophas, and M. van den Brand, "Painting the landscape of automotive software in GitHub," in *Proceedings of the 19th International Conference on Mining Software Repositories*. ACM, May 2022.

[9] S. Alcaide, G. Cabo, F. Bas, P. Benedicte, F. Fuentes, F. Chang, I. Lasfar, R. Canal, and J. Abella, "SafeX: Open Source Hardware and Software Components for Safety-Critical Systems," in *2022 Forum on Specification & Design Languages (FDL)*. IEEE, Sep. 2022.

[10] S. M. Sulaman, A. Orucevic-Alagic, M. Borg, K. Wnuk, M. Host, and J. L. de la Vara, "Development of Safety-Critical Software Systems Using Open Source Software – A Systematic Map," in *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, Aug. 2014.

[11] M. Cinque, L. D. Simone, and A. Marchetta, "Certify the Uncertified: Towards Assessment of Virtualization for Mixed-criticality in the Automotive Domain," in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, Jun. 2022.

[12] Z. Assaad, N. Derwort, and K. A. Daniell, "Considerations for Assuring Software Systems of Autonomous Aircraft," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, Dec. 2020.

[13] H. Espinoza, A. Ruiz, M. Sabetzadeh, and P. Panaroni, "Challenges for an Open and Evolutionary Approach to Safety Assurance and Certification of Safety-Critical Systems," in *2011 First International Workshop on Software Certification*, 2011, pp. 1–6.

[14] D. Streitferdt, A. Zimmermann, J. Schaffner, and M. Kallenbach, "Component-wise software certification for safety-critical embedded devices," in *2017 8th Annual Industrial Automation and Electromechanical Engineering Conference (IEMECON)*, 2017, pp. 175–180.

[15] A. Bilbao, I. Yarza, J. L. Montero, M. Azkarate-askasua, and N. Gonzalez, "A railway safety and security concept for low-power mixed-criticality systems," in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, 2017, pp. 59–64.

[16] R. Pietrantuono and S. Russo, "Robotics Software Engineering and Certification: Issues and Challenges," in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2018, pp. 308–312.

[17] H. Bagheri, E. Kang, and N. Mansoor, "Synthesis of Assurance Cases for Software Certification," in *2020 IEEE/ACM 42nd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, 2020, pp. 61–64.

[18] N. Hrgarek, "Certification and regulatory challenges in medical device software development," in *2012 4th International Workshop on Software Engineering in Health Care (SEHC)*, 2012, pp. 40–43.

[19] R. Adler, S. Kemmann, D. de Melo Carvalho Filho, and J. A. O. Neto, "Safety assessment of software-intensive medical devices: Introducing a safety quality model approach," in *2013 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2013, pp. 217–222.

[20] P. Stirgwolt, "Effective management of functional safety for ISO 26262 standard," in *2013 Proceedings Annual Reliability and Maintainability Symposium (RAMS)*. IEEE, Jan. 2013.

[21] K. Wallace, "Safe and secure: re-engineering a software process set for the challenges of the 21st century," in *9th IET International Conference on System Safety and Cyber Security (2014)*. Institution of Engineering and Technology, 2014.

[22] R. Palin, D. Ward, I. Habli, and R. Rivett, "ISO 26262 safety cases: compliance and assurance," in *6th IET International Conference on System Safety 2011*. IET, 2011.

[23] S. Wagner, B. Schatz, S. Puchner, and P. Kock, "A Case Study on Safety Cases in the Automotive Domain: Modules, Patterns, and Models," in *2010 IEEE 21st International Symposium on Software Reliability Engineering*. IEEE, Nov. 2010.

[24] P. Jurnecka, M. Barabas, P. Hancek, M. Henzl, and M. Kacic, "A method for parallel software refactoring for safety standards compliance," in *8th IET International System Safety Conference incorporating the Cyber Security Conference 2013*. Institution of Engineering and Technology, 2013.

[25] B. Chimdyalwar, A. Jana, S. Kumar, A. Khadsare, and V. Ghime, "Identifying Relevant Changes for Incremental Verification of Evolving Software Systems," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, Mar. 2022.

[26] A. Agrawal, S. Khoshmanesh, M. Vierhauser, M. Rahimi, J. Cleland-Huang, and R. Lutz, "Leveraging Artifact Trees to Evolve and Reuse Safety Cases," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, May 2019.

[27] M. Li, B. Meng, H. Yu, K. Siu, M. Durling, D. Russell, C. McMillan, M. Smith, M. Stephens, and S. Thomson, "Requirements-based Automated Test Generation for Safety Critical Software," in *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*. IEEE, Sep. 2019.

[28] H. Zheng, J. Feng, W. Miao, and G. Pu, "Generating Test Cases from Requirements: A Case Study in Railway Control System Domain," in *2021 International Symposium on Theoretical Aspects of Software Engineering (TASE)*. IEEE, Aug. 2021.

[29] S. Sheikhi, E. Kim, P. S. Duggirala, and S. Bak, "Coverage-Guided Fuzz Testing for Cyber-Physical Systems," in *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, May 2022.

[30] H. Y. Yun, Y. J. Joe, and D. M. Shin, "Method of license compliance of open source software governance," in *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2017, pp. 83–86.