Proceedings of the 35th European Safety and Reliability & the 33rd Society for Risk Analysis Europe Conference
Edited by Eirik Bjorheim Abrahamsen, Terje Aven, Frederic Bouder, Roger Flage, Marja Ylönen
 Published by Research Publishing, Singapore.
doi: 10.3850/978-981-94-3281-3_ESREL-SRA-E2025-P1355-cd

RAMS is not enough!
The design of a software integration risk analysis matrix (SIRAM) for assessing the impact of software integration on physical system performance

Arno Kok

Asset Management and Maintenance Engineering, University of Twente, The Netherlands, a.t.kok@utwente.nl
Integration and Train Performance, NS Treinmodernisering Engineering, The Netherlands

Alberto Martinetti

Asset Management and Maintenance Engineering, University of Twente, The Netherlands, a.martinetti@utwente.nl

Ferry Verrijzer

Integration and Train Performance, NS Treinmodernisering Engineering, The Netherlands, ferry.verrijzer@ns.nl

Jan Braaksma

Asset Management and Maintenance Engineering, University of Twente, The Netherlands, a.j.j.braaksma@utwente.nl

Physical assets are increasingly digitally enhanced using software and associated information technologies. These added functionalities make them more complex to engineer and maintain as they depend on additional software and information technology components. However, the available RAMS analysis methodologies do not explicitly include software application and integration. We propose a software integration risk matrix (SIRAM) as an extension of the current RAMS methodology, to assess the effect of software on overall system performance. This extension can aid decision-making by indicating the expected software's integration impact on system performance and maintenance needs and is developed using design science research methodology (DSRM). A case study within the Dutch railways served as the basis for the design and testing of the proposed matrix. The testing shows that the proposed software integration risk matrix can add value by managing that critical software impacts will be part of the system integration process.

Keywords: RAMS, Software, Information Technology, System Integration, IT/OT convergence, Railways.

1. Introduction

The software increasingly influences almost all technical systems and modern physical assets. Reliability, availability, maintainability and safety (RAMS) analyses are performed to ensure, amongst others, the reliability and maintenance requirements of new or modified systems. These RAMS analysis methodologies are perceived to be effective for traditional electromechanical systems. However, traditional RAMS methods do not specifically pay attention to the unique aspects of software integration. Our previous study shows that the RAMS methodology has difficulties representing the reliability and maintenance effort required for modern-day systems (Kok et al., 2023).

Until the late 20th century, assets consisted mainly of electromechanical components; today, they are increasingly equipped with IT software and hardware. These electromechanical systems often fail independently, and defects are predictable, while software failures are usually random events (April & Abran, 2008; Lyu, 2007). Furthermore, unlike physical systems, software is

unaffected by environmental factors and does not deteriorate over time (Naylor & Joyner, 2014). However, the software can have ageing symptoms due to an accumulation of memory errors, for example. This results in increased failure rates and/or performance degradation. Unlike physical systems, restarting the application often solves software ageing problems (Araujo et al., 2021). Also, it is generally not possible to define a failure rate for software (Singpurwalla, 1995). Nonetheless, software defects can cause poor system performance and downtime (Eloff & Bella, 2018). The effect of software failures on performance is generally between 20-50% (Gray, 1985; Schroeder & Gibson, 2006), but sometimes up to 2/3 of failures can be related to software (Grottke et al., 2010).

Some examples of these software defects are two rolling stock and a metro failure in the Netherlands that made it to the (regional) news. These failures that impacted passengers resulted from software and hardware interactions, leading to hidden and unclear failure modes (Bremmer, 2024; NOS, 2022; NU.nl, 2019).

From an internal investigation within the Dutch Railways (Peters, 2024) based on maintenance service request data, the impact of software-related failures on corrective maintenance activities is between 8-13%, depending on the type of train and the position within the lifecycle of this train.

These figures might even be higher since about 20% of the service requests have no registered cause, and the mechanic did not find an error. These failures that "come and go" are likely related to software failures, as hardware failures often have a mechanical cause that does not spontaneously disappear.

A software defect can refer to both a fault (cause) and a failure (effect) (Lyu, 1996). A software failure results from a software fault originating from a software code defect (Musa, 2004). In the performance of assets, there is an interplay between hardware, software and humans, and it is often difficult to find the exact cause of a failure. Therefore, we will use "software-related failures" to refer to failures arising from hardware, software and human interaction within assets.

1.1. Challenges when integrating software components within physical systems

Assets that contain software face a unique issue due to the relationship between hardware and software, where even a small error can cause significant failures (Oveisi & Ravanmehr, 2017). These failures often occur at the interfaces between systems (Carlson, 2012), and the interfaces are hidden within the code, making fault-finding even more challenging (González-Arechavala et al., 2010).

Consequently, software fault prevention during software development is essential. Despite this, software fault prevention techniques have not kept up with the increased complexity of software (Goble, 2010), and all faults and failures discovered during the software development process affect software reliability (Naylor & Joyner, 2014). During operation, other aspects influence the reliability of the software components, such as operators, internal or external hardware, or any combination of these (Goble, 2010). Therefore, software verification and validation are crucial (Björklund et al., 2021). Moreover, to keep digitized assets secure, software updates are necessary to fix known software-related issues such as vulnerabilities and bugs. Within the literature, there are different opinions on how to classify these software-related issues. In Table 1, an overview of several software-related issue classifications is given. This overview shows that software-related problems do not only originate from so-called bugs in the source code but can have multiple other sources, such as configuration or network-related issues.

Table 1. Different software-related issue classifications adapted from (Catolino et al., 2019; Herzig et al., 2013; IEC, 2012; Gladney, 2007).

Author	List of classifications
Catolino et al.	Configuration issue
	Network issue
	Database-related issue
	GUI-related issue
	Performance issue
	Permission/depreciation issue
	Security issue
	Program anomaly issue
	Test code-related issue
Herzig et al.	Bugs
	Feature requests
	Documentation issue

	Refactoring request
	Improvement request
	Other
Gladney	Media and hardware failures
	Software failures
	Communication channel errors
	Network service failures
	Component obsolescence
	Operator errors
	Natural disasters
	External attacks
	Internal attacks
CENELEC 62628:2012	Specification fault
	Design fault
	Programming fault
	Compiler-inserted fault
	Faults introduced during maintenance

1.2.Current Approaches and Gaps

Until now, most software reliability research has focused on enhancing software resilience by preventing or removing software faults (i.e., software process improvement) (Goble, 2010). Quality control during software development is essential since software defects are all quality aspects (Jelinski & Moranda, 1972). The level of confidence in a software development process is often called assurance. This software quality assurance focuses on testing practices, requirements validation, used technologies, software change management controls, and organization and culture (Al MohamadSaleh & Alzahrani, 2023; IEC, 2012). However, justifying the correlation between process quality and the failure rate of the software product is nearly impossible (Habli et al., 2010).

There are many software reliability prediction techniques available. However, there are challenges when these techniques are applied in practice (Oveisi & Ravanmehr, 2017). The difficulty with these methods is that they require in-depth knowledge of the software and often need the actual software code. System integrators usually do not have either (Rathi et al., 2022). No model is available to combine the effects of software reliability and that of the hardware. This results in too little attention to the influence of software on operational performance and expected maintenance efforts during the design and integration of systems within assets. Therefore, our study is focused on designing an extension of the existing RAMS methodology to evaluate the impact of software on reliability and maintenance when integrating systems within an asset.

2. Methodology

This research presents an extension of the RAMS methodology developed using design science research methodology (DSRM) (Peffers et al., 2007). In Fig. 1, this process is depicted.

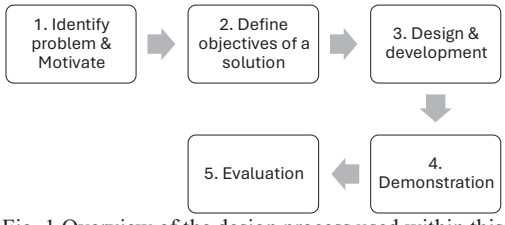

Fig. 1 Overview of the design process used within this study, adapted from (Peffers et al., 2007).

2.1.Identify problem and motivation

The primary issue is that software has an increasing impact on the reliability of the systems within an asset and, thus, the reliability and maintenance performance of the asset itself. However, the current RAMS method does not include software (Kok et al., 2023).

2.2. Design objectives

The purpose of the proposed software integration risk analysis matrix is for engineers to assess the effect of the integration of software on the performance of their system, and it is based on the main observations from our previous studies (Kok et al., 2023). We concluded, amongst others, that the current RAMS methodology underestimates software's impact on system reliability, overlooks the effects of IT network performance, lacks multidisciplinary expertise, and requires an iterative approach for optimizing system performance. These challenges have led to the following design objectives:

- Objective 1: Identifying when software integration within an asset is critical and thus requires extra engineering attention.
- Objective 2: Adding attention to the specific effects of software on the reliability prediction.

- Objective 3: An easy-to-use model: a hardware engineer should be able to easily assess the identify and assess the impact of software on reliability using the extension of the method.

2.3.Co-designing the extension of the method

The extension of the method is developed in active involvement with industry experts from the NS Engineering department in Haarlem, The Netherlands. The principal researcher laid out the problem in multiple sessions and brainstormed possible solutions with the participants.

2.4.Demonstration of the method

The results of these sessions were presented to 18 experts to get an estimate of the usability of the proposed method. During a semi-structured interview, using the questions from Appendix A, these experts were asked how useful the method and accompanying matrix are.

2.5.Evaluation of the method

The validity of the matrix is assessed by analyzing the interviews with the railway industry experts using two steps. In the first step the outcomes of these interviews will be scanned and mapped to the papers design objective, to see what is emerging. In the second step two interviews are selected and used as a first validation.

3. Results

A RAMS calculation must be prepared for the electromechanical part of the system under consideration according to the existing methodology that is based on EN50126 (NEN, 2017). From this RAMS calculation a failure rate is determined, usually in different failure classes (FC1, FC2, FC3) and expressed in failures per million coach kilometers (FMCK).

Then for identifying and assessing the impact of software on system performance a matrix with accompanying instructions is proposed, objective 1 and 2. After following several steps a factor can be read from this SIRAM matrix. This factor can be used to adjust the earlier calculated failure rate of the electromechanical part of the.

The matrix which should be easy-to-use, objective 3, is inspired by several studies on managing the impacts of software on system performance (Chittister & Haimes, 1996; Ye & Kelly, 2004; Smith, 2004; Roca, 2019). Ideally, one would derive the factors in the matrix from actual generalized failure rates for software. In practice, this is not possible, as substantiated in the introduction to this article.

The proposed matrix consists of two axes: how software intensive is the design (e.g. how much impact does the software have on the system it is controlling), and what is the complexity level of the system (e.g. a system with standard software and hardware with limited interfaces or is it specifically created for the situation at hand with many hard- and software interfaces).

Table 2. The proposed SIRAM matrix will determine the level of attention that should be given to software development.

Determine multiplication factor fs		Impact on system performance			
		Low	Medium	High	Very High
Complexity level	Low	1,05	1,10	1,20	1,30
	Med	1,10	1,20	1,30	1,40
	High	1,20	1,30	1,40	1,50

The matrix's goal is to indicate for practitioners how important it is to consider the impact of the software on the system reliability and the required maintenance effort. The method consists of three steps to determine the influence of the software that is being considered.

First, the impact of software within a system is determined using Table 3. Second, the complexity level of the software within a system is determined in Table 4. Third, using the results of these tables, the software's influence on the system's performance under consideration can be read from the matrix, see Table 4. The given numbers are a multiplying factor to adjust the calculated reliability of a system for the influence of software. The colors correspond with the additional effort needed to ensure the robustness of the software.

Table 3. Matrix to determine the impact of the software on the system.

System complexity level		
Level	**Description**	**Example**
Low	Standardized software, limited in size	Software for intelligent light control
Medium	Standardized software, but big in size. Or custom software, but limited in size	Intelligent climate system with heat pump and CO2 level control
High	Custom software and big in size	TCMS update

Table 4. Matrix to determine the complexity of the software within a system.

Impact on system performance		
Level	**Description**	**Example**
Low	Software failure barely affects the system	System continues to function, but status report no longer works
Medium	Software failure interferes with system operation, but system remains available	The brightness controller fails, leaving the lamps at maximum brightness without adjusting for conditions.
High	Software failure severely limits system functionality	
Very high	Due to software failure, the system no longer functions	NVR fails, resulting in no more camera images being saved

Demonstration

In preparation for the interview, the experts were asked to read the method and complete the matrix with one example system in mind. This system was either the European Train Control System (ETCS) or a Closed-Circuit Television (CCTV) system. Since not all participants have experience with both systems, we have divided the interviews into CCTV or ETCS, depending on their experience. In the instructions that were shared with the experts before the interview, the matrix was made gray, and the factors were removed to prevent bias. During the interview, the experts are asked to plot the assigned system in the matrix and then discuss their reasoning before we share the matrix with colors and factors

4. Discussion

In this section, the initial observations are mapped to the paper's design objectives and outcomes of two interviews on CCTV will be elaborated upon, and some. After scanning the interview results the preliminary insights from the interviews are mapped to the, in paragraph 2.2 outlined, design objectives of this paper in Table 6.

The first objective of the paper is to identify when software development during system integration needs extra attention. The matrix does not accurately represent the quality of the software development process, it is suggested to classify the systems within the matrix instead of ranking them.

The paper's second objective is to add the effects of software to the reliability prediction. The methodology effectively raises awareness about software quality and its impact on asset reliability, but it may not always be accurate. Software is often hidden from most people. By using a factor, the impact of software becomes measurable. Making software visible helps in the conversations with industry partners.

The third objective of the paper is to design a method that is easy to use. The first results show that the method is indeed easy to use; however, one of the respondents indicated that the matrix might be too simple.

Then the high-level results of the two selected interviews are presented in Table 5. As can be seen from this table, both experts classify the CCTV system in the same impact and complexity categories. Also, both experts indicate that colors can be used to determine the effort needed for the software being evaluated.

Table 5. Impact and complexity classification by two experts on CCTV.

Category	Respondent #13	Respondent #16
Impact	Very High	Very High
Remark	Many dependencies around software across components.	Tricky because various software parts have different impacts.
Complexity	Average	Average
Is the method helpful?	Yes, however, using only colors instead of both colors and numbers is preferred.	Only when using colors not when using numbers.
Remarks		The complexity is low with standard systems and higher on custom systems.

Last, some general observations based on the interviews. There seems to be a difference between the people who have read the method's instructions and those who had not and who had read them were more effective in its use. Which suggests it is needed to train or facilitate the practitioners in the use of the matrix.

When the method is used with only numbers, the importance of a system on the asset's overall performance is fixed within the allocation of the reliability budget for the mechanical part of the systems. If only color categories are used, to express the criticality, this allocation is not present; therefore, additionally, the system impact on the overall train reliability needs to be

considered. The next step is to extend the validation of the method using all the interview data and to improve the method based on the feedback from the interviews. Then, the improved matrix should be validated within other industrial domains.

Table 6. Feedback on the proposed method mapped to the objectives of this paper.

Objectives	Feedback
1: Identifying when software integration within an asset is critical and thus requires extra engineering attention.	The numbers in the proposed matrix are difficult. Instead of ranking a system, classifying the systems generically beforehand might work better. The matrix does not necessarily visualize good or bad software development.
2: Adding the effects of software to the reliability prediction	The methodology is useful. This may trigger more attention to software quality and reliability. It is useful as it raises awareness of the software's importance at the start of the project.
3: Easy-to-use, a hardware engineer should be able to assess the impact of software on reliability using the extension of the method.	The matrix makes the discussion on the impact accessible. The risk matrix might however be too simplistic. If you work with colors, people will perhaps only act if they are really in red, for instance.

5. Conclusion

Physical assets are becoming more digitally complex, relying heavily on software and IT components. However, software reliability is often excluded from RAMS analysis and cannot easily be quantified regarding expected failure rates. This study proposes extending the RAMS methodology with a decision-making matrix to assess software's impact on overall asset reliability. This matrix helps engineers to determine the importance of software in the reliability of the system at hand and to focus their RAMS analysis efforts effectively. The next step is to improve the method based on the feedback, and to validate this improved matrix in other industrial domains.

Acknowledgements
The first author would like to thank the experts from the Nederlandse Spoorwegen for participating in the interviews and his fellow PhDs from the AMME group for their support during the creation of this paper. Holland High Tech and Nederlandse Spoorwegen (NS) supported this work with a PPP grant for research and development in the top sector HTSM.

Appendix A. Structured interview questions
Interviewee
Interviewer
Date
Location
Was the matrix documentation reviewed?
RAMLCC experience?
Level of railway-related software experience?
Is the document understandable?
What example system is taken in mind?

Complexity:
Is the complexity of your system determinable?
Are the categories logical/applicable?
Missing categories?

Impact:
Determinable?
Logical /applicable?
Missing categories?

Matrix:
Which position emerged?
What would be real value?
Does this match the number in the completed matrix?
Discuss deviation

Response to the pre-filled matrix:
Does the methodology make sense?
Do you prefer numbers or colors?
Other comments/suggestions?

6. References

Al MohamadSaleh, A., & Alzahrani, S. (2023). Development of a Maturity Model for Software Quality Assurance Practices. *Systems*, *11*(9), Article 9. https://doi.org/10.3390/systems11090464

April, A., & Abran, A. (2008). *Software maintenance management: Evaluation and continuous improvement*. Wiley Interscience. https://ieeexplore.ieee.org/book/6129685

Araujo, J., Melo, C., Oliveira, F., Pereira, P., & Matos, R. (2021). A Software Maintenance Methodology: An Approach Applied to Software Aging. *2021 IEEE International Systems Conference (SysCon)*, 1–8. https://doi.org/10.1109/SysCon48628.2021.9447082

Björklund, L., Glaser, M., Skofteland, G., & Lundteigen, M. A. (2021). A Comparison of Different Approaches for Verification and Validation of Software in Safety-Critical Systems. *Proceedings of the 31st European Safety and Reliability Conference (ESREL 2021)*, 3451–3458. https://doi.org/10.3850/978-981-18-2016-8_262-cd

Bremmer, D. (2024, February 18). NS-paradepaardje 'de wesp' blijkt hoofdpijntrein: 99 treinen besteld, slechts acht daarvan rijden. *AD.Nl/Algemeen Dagblad*. https://www.ad.nl/binnenland/ns-paradepaardje-de-wesp-blijkt-hoofdpijntrein-99-treinen-besteld-slechts-acht-daarvan-rijden~a730a56a/

Carlson, C. (2012). *Effective FMEAs: Achieving safe, reliable, and economical products and processes using failure mode and effects analysis*. Wiley. https://onlinelibrary.wiley.com/doi/book/10.1002/9781118312575

Catolino, G., Palomba, F., Zaidman, A., & Ferrucci, F. (2019). Not all bugs are the same: Understanding, characterizing, and classifying bug types. *Journal of Systems and Software*, *152*, 165–181. https://doi.org/10.1016/j.jss.2019.03.002

Chittister, C. G., & Haimes, Y. Y. (1996). Systems Integratilon via Software Risk Management. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, *26*(5), 521–532. https://doi-org.ezproxy2.utwente.nl/10.1109/3468.531900

Eloff, J., & Bella, M. B. (2018). Software Failures: An Overview. In J. Eloff & M. Bihina Bella (Eds.), *Software Failure Investigation: A Near-Miss Analysis Approach* (pp. 7–24). Springer International Publishing. https://doi.org/10.1007/978-3-319-61334-5_2

Gladney, H. M. (2007). *Preserving digital information*. Springer Berlin, Heidelberg. https://link.springer.com/book/10.1007/978-3-540-37887-7

Goble, W. (2010). *Control Systems Safety Evaluation and Reliability, Third Edition* (3rd ed). International Society of Automation.

González-Arechavala, Y., Rodríguez-Mondéjar, J. A., & Latorre-Lario, G. (2010). The opportunity to improve software RAMS. In B. Ning (Ed.), *WIT Transactions on State of the Art in Science and Engineering* (1st ed., Vol. 1, pp. 91–102). WIT Press. https://doi.org/10.2495/978-1-84564-494-9/11

Gray, J. (1985). *Why Do Computers Stop and What Can Be Done About It?* (85.7). Tandem Computers. https://pages.cs.wisc.edu/~remzi/Classes/739/Fall2018/Papers/gray85-easy.pdf

Grottke, M., Nikora, A. P., & Trivedi, K. S. (2010). An empirical investigation of fault types in space mission system software. *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, 447–456. https://doi.org/10.1109/DSN.2010.5544284

Habli, I., Hawkins, R., & Kelly, T. (2010). Software safety: Relating software assurance and software integrity. *International Journal of Critical Computer-Based Systems*, *1*(4), 364. https://doi.org/10.1504/IJCCBS.2010.036605

Herzig, K., Just, S., & Zeller, A. (2013). It's not a bug, it's a feature: How misclassification impacts bug prediction. *2013 35th International Conference on Software Engineering (ICSE)*, 392–401. https://doi.org/10.1109/ICSE.2013.6606585

IEC. (2012). *IEC 62628:2012—Guidance on software aspects of dependability* (62628; Version 2012). Nederlands Normalisatie-instituut. https://connect.nen.nl/Standard/Detail/176136?compId=16755&collectionId=0

Jelinski, Z., & Moranda, P. (1972). Software Reliability Research. In *Statistical Computer Performance Evaluation* (pp. 465–484). Elsevier. https://doi.org/10.1016/B978-0-12-266950-7.50028-1

Kok, A., Martinetti, A., & Braaksma, J. (2023). RAMS never dies! Applying the approach to IT/OT converged systems. *Proceedings of the 33rd European Safety and Reliability Conference (ESREL 2023)*, 3181–3187. https://doi.org/10.3850/978-981-18-8071-1_P286-cd

Lyu, M. R. (Ed.). (1996). *Handbook of software reliability engineering*. IEEE Computer Society Press ; McGraw Hill. https://www.cse.cuhk.edu.hk/~lyu/book/reliability/

Lyu, M. R. (2007). Software Reliability Engineering: A Roadmap. *Future of Software Engineering (FOSE '07)*, 153–170. https://doi.org/10.1109/FOSE.2007.24

Musa, J. D. (2004). *Software reliability engineering: More reliable software, faster and cheaper* (2nd ed). AuthorHouse. https://dl.acm.org/doi/10.5555/1036241

Naylor, W., & Joyner, B. (2014). A discourse on software safety and software reliability. *2014 Reliability and Maintainability Symposium*, 1–5. https://doi.org/10.1109/RAMS.2014.6798493

NEN. (2017). *EN 50126—1—Railway Applications—The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS)—Part 1: Generic RAMS Process* (50126–1). NEN.

NOS. (2022, January 25). *"Structurele problemen door nieuw metrosysteem in Amsterdam."* https://nos.nl/l/2413149

NU.nl, D. (2019, February 20). *"Nieuwe sprinter van NS kampt met storingen en valt vaak uit."* NU. https://www.nu.nl/binnenland/5752206/nieuwe-sprinter-van-ns-kampt-met-storingen-en-valt-vaak-uit.html

Oveisi, S., & Ravanmehr, R. (2017). Analysis of software safety and reliability methods in cyber physical systems. *International Journal of Critical Infrastructures*, *13*(1), 1. https://doi.org/10.1504/IJCIS.2017.083632

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Peters, L. (2024, September 2). *Software gerelateerde storingen Nieuw Materieel—Maximo Analyse*. NS Treinmodernisering. 850110750 (internal document)

Rathi, G., Tiwari, U. K., & Singh, N. (2022). Software Reliability: Elements, Approaches and Challenges. *2022 International Conference on Advances in Computing, Communication and Materials (ICACCM)*, 1–5. https://doi.org/10.1109/ICACCM56405.2022.10009422

Roca, J. L. (2019). *Software Reliability: A Must*. https://www.researchgate.net/publication/337289747_Software_Reliability_A_Must

Schroeder, B., & Gibson, G. A. (2006). A large-scale study of failures in high-performance computing systems. *International Conference on Dependable Systems and Networks (DSN'06)*, 249–258. https://doi.org/10.1109/DSN.2006.5

Singpurwalla, N. D. (1995). The failure rate of software: Does it exist? *IEEE Transactions on Reliability*, *44*(3), 463–469. https://doi.org/10.1109/24.406582

Smith, J. D. (2004). ImpACT: An Alternative to Technology Readiness Levels for Commercial-Off-The-Shelf (COTS) Software. In R. Kazman & D. Port (Eds.), *COTS-Based Software Systems* (Vol. 2959, pp. 127–136). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-24645-9_24

Ye, F., & Kelly, T. (2004, August 2). Criticality Analysis for COTS Software Components. *Proceedings of 22nd International System Safety Conference (ISSC'04)*. 22nd International System Safety Conference, Providence, Rhode Island, USA. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=85fbaff51b0b3bc4fd8060c9d77c551de017bfd9