Security Related Technical Debt in the Cyber-Physical Production Systems Engineering Process

Bernhard Brenner, Edgar Weippl
Christian Doppler Laboratory
for Security and Quality Improvement
in the Production System Lifecycle (CDL-SQI)
Institute of Information Systems Engineering
TU Wien
[first].[last]@tuwien.ac.at

Andreas Ekelhart
SBA Research
Vienna, Austria
aekelhart@sba-research.org

***Abstract*—Technical debt is an analogy introduced in 1992 by Cunningham to help explain how intentional decisions not to follow a gold standard or best practice in order to save time or effort during creation of software can later on lead to a product of lower quality in terms of product quality itself, reliability, maintainability or extensibility. Little work has been done so far that applies this analogy to cyber physical (production) systems (CP(P)S). Also there is only little work that uses this analogy for security related issues. This work aims to fill this gap: We want to find out which security related symptoms within the field of cyber physical production systems can be traced back to TD items during all phases, from requirements and design down to maintenance and operation. This work shall support experts from the field by being a first step in exploring the relationship between not following security best practices and concrete increase of costs due to TD as consequence.**

***Index Terms*—Technical Debt, Technical Debt in the context of Security, Cyber Physical Production Systems**

I. Introduction

The introduction of so-called cyber physical production systems or CPPS brings many benefits, such as a gain of efficiency for manufacturers [1]. However, the increased reliance on complex software, the all-time connectedness (between machines or even from machine to the Internet/a data cloud) using established Internet Protocol (IP) technology also introduces a considerable amount of new attack vectors into the field of production system engineering. While the arising threats are generally similar to the ones already known from classical computer security [2], [3], the consequences of successful attacks are about to approach the physical space.

Security is a key issue here and following established security concepts is thus very important. Due to the similarities, many of the threats, vulnerabilities and good/bad practices are already known from the fields of software engineering and software security and such best practices have been introduced already more than a decade ago [4]–[6].

One of these good practices, lent from the field of software development, are the "seven touchpoints of software security" introduced in 2004 by [4], [7], [8].

In this work, we take these touchpoints as best-practice gold standard. We adapt them slightly regarding the creation of CPS and map them to the CPS planning phases (Section III). We then explore TD items, causes and products in CPPS that are related to security and explore which of them can be related to not following a common secure-by-design best practice (Section IV).

As software and CPS are both technical systems, the similarities are high and we strongly believe that the concept of TD is similarly relevant.

II. Definitions

The term *technical debt* (TD) has its origins in the field of software engineering. It has been introduced by Cunningham in 1992 as a metaphor to help explain the need of software code refactoring and other improvements to nontechnical stakeholders [9].

In principle, technical debt describes the (maybe accumulative) deviation from best practices and their possible consequences often aiming to keep a certain schedule or effort savings, but, in most cases, having negative consequences regarding effort or schedule on the long run [10].

When defining TD for the context of this paper, we also want to refer to the graph and description of Kruchten, Nord and Oszkaya presented in 2012. In his work, he differentiates between TD in terms of quality issues and TD in terms of the lack of new features [11]. In our paper, we want to focus on TD in terms of quality issues.

Within this context, Vogel-Heuser et al. point out that due to the long maintenance phase of production systems of up to 50 years, it is likely that a long phase in which interest has to be paid will follow if TD is aquired during the creational phases of a CPS [12]. This is the reason why we chose the seven touch points by mcgraw as role-model concept: It is a

978-1-7281-4878-6/19/$31.00 ©2019 IEEE

proven concept to design secure-by-design systems, and has the potential to save costs during the operational phase of a cps which would otherwise be incurred during the (very long) operational phase.

Within the context of TD we define four basic elements in the context of this paper.

A. TD Entities

- **Technical debt items** are one or more concrete tangible artifacts of a system or a creation process, e.g.: code or components, documentation, tests.
- **TD causes** are the causes of a certain TD, they can be a process or action, the lack of an action, an event that triggers TDs existence, e.g., the loss of a key person.
- **TD consequences** are either some kind of lowering in value or quality of the system/product or increased burden, such as schedule pressure or increased necessary maintenance/extension effort.
- **TD products** are the final consequences of a certain kind of TD. The goal of the product or system can be affected through delays, the lowering of quality or any kind of difficulty to keep the system or process running.

Let's provide a basic TD example: Imagine a software project that is planned to be done within six months by two people. One of them becomes ill and can't work for two months, thus there is a lack of approximately 17% of the total effort in person-hours.

The client, however is not willing to accept any delay and therefore the two developers decide to leave out some time-intensive testing and documentation of the code.

At time of project handover, the client is satisfied with the product and the software is sold to the client. However, a few months later, the client complains about a bug and wants the software company to repair it. This procedure is time intensive and effort taking, especially due to the lack of documentation and the fact that they haven't been looking at this code for months, but they eventually manage to fix it and send it to the client. The client, however, is not as satisfied with the product as before and loses some trust into the company.

Clearly, fixing the bug afterwards required a lot more effort - not only but also due to the lack of documentation. This bug would maybe never have arisen if they did not cut on testing efforts beforehand and the client would probably have more trust into the company now if they did not have had to fix any problems after shipping. Still, they had to invest time to fix the bug even though they tried to save this time before. In addition, fixing the bug afterwards took them even longer since they had to remember their own code and since they had to find it, despite the lack of proper documentation. And even worse, the same applies for any future bug that arises. Thus, the consequences of the decision for the quick win in the past are not only more expensive than the savings in terms of time, but may also multiply in the future.

Therefore, we can say that the software team acquired TD during the development phase, which they later have to pay back during maintenance and services and that the payback is a lot higher.

Even though the concept in principle comes from the field of software engineering [10], [11], it can be adapted to match other fields as well and also to the field of cyber physical production systems [12]–[15].

Within this context, we will use the terminology "product" for the cyber physical production system that is to be designed (e.g. a production line producing gearbox assemblies, such as shift forks) and the final product (or endproduct), which is the actual sellable product the cyber physical system shall produce (e.g. a shift fork).

III. The Seven Touchpoints as Standard for CPS

The following section describes how we are going to use the seven touchpoints as best practice concept for developing secure-by-design production systems. We assume the construction process for cyber physical production systems to follow the following order:

- Define goal
- Rough system structure
- Mechanical engineering
- Electrical engineering
- Control software
- Commissioning
- Operation and maintenance

Figure 1 shows how we put them into the structure of the seven touchpoints. The original touchpoints are described in [4]. This figure has been motivated by the original from software engineering and modified for CPS engineering [12]. In the center of the graph (blue bar with arrow blocks), there are the seven touchpoints with slightly abstracted terms (e.g. system instead of software), formulated as results that shall be achieved in the according planning phase. The orange blocks describe tasks that have to be performed to achieve these results. The color bars below denote ten typical design phases of cyber physical production systems. The position of these bars is arranged so that they fit to where the results within the blue blocks should be achieved.

During definition of the system's objectives, the use case design including the abuse cases as well as the definition of requirements for the system, including the security requirements shall be performed. It shall be checked if these requirements are fulfilled and the abuse cases are countermeasured already when designing the rough CPS structure and defining the system objectives.

The electrical plans should undergo comprehensive security tests and system analysis. More precisely, independent experts should have a detailed look at the engineers' output, prior to hand in. The same is valid for the mechanical plans.

Once the CPS is set up, it shall undergo comprehensive system tests and penetration tests, which in the case of cyber physical production systems could be the following:

- Testing the countermeasures of the abuse cases by trying to perform them (either on real systems if applicable or within test settings)

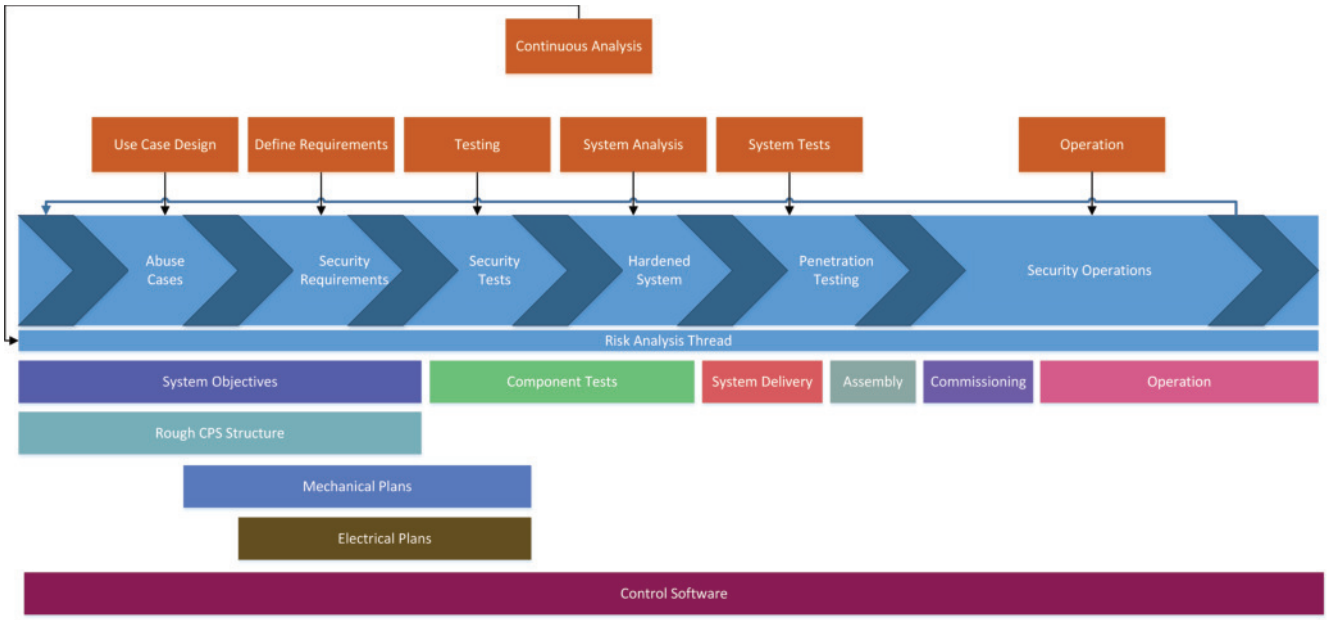

Fig. 1. The seven touchpoints of software security as reference during the design of cyber physical production systems.

- Auditors may try to break in or even to trigger a stall in operation/production. In order to fulfill safety requirements, this test can also be performed on simulated systems such as digital twins [16].

The *CPS-equivalent* to penetration testing in software systems is its logical equivalent, namely to test the system's resistance against certain types of mechanical/physical or also electrical-, electromagnetic or logical active & passive attacks on either sensors, actors, or the control electronic, or even the control software. Vulnerabilities that have been found shall then be reported to the responsible constructor or manufacturer, and the responsible engineers shall decide if and how to cope with or fix them. And, of course control logic and control software may be tested the same way as conventional software.

The project phase *control software* (purple bar) covers all seven *touchpoints*. For software (including control software), however, they can be used in an unchanged manner. However, the control software should be created within an independent project including its own requirements, abuse cases, etc.

Lastly, there is one task which covers all of the planning- and the operation-phase, the *risk analysis thread*, which shall demonstrate that continuous risk analysis during all project phases is necessary.

TD can arise due to a deviation from these *seven touchpoints*. Below are some examples:

If time has been saved upon making abuse cases at the beginning of the system objectives and the system design, they may later on (during operation) become an obvious case that is not covered and thus there later on is a threat that endangers the continuous operation of the cyber physical system.

The endproduct may be susceptible to electrical attacks such as glitching [17, p. 59] since no one ever thought about or time has been saved upon security tests during the creation of the electrical plans of the final product.

If continuous risk analysis is not performed, for example, decisions upon new risks arising (e.g. a new generation of malware) could be neglected. This also leads to vulnerabilities of the system and may endanger important assets (e.g. the availability of the production system due to standstills) as a consequence.

Although a lot is similar, there are in fact differences between software systems and cyber-physical production systems considering the security aspects: assuming that a cyber-physical production system runs a certain control software and is connected to the Internet, it offers all attack vectors a typical software system running on standard PC hardware has. However, due to its operational technology, and thus physical, nature, it also offers a wide range of additional attack vectors.

IV. How Deviations from the Security Gold Standard can lead to TD

Now we want to take a look at where deviations can arise and how TD could emerge. In the following, we will list a selection of TD products and analyze their causes. Our approach is explained within the explanation of the first TD example case in Section IV.

TD Product: Standstill of the production. A standstill of the production within this context is any kind of time span, in which the production line is (unwantedly) not available.

Figure 2 shows the according diagram, which attempts to visualize cause and effect relationships within a tree structure. The diagram shall be read as follows: "production standstill" is the problem or TD product. Below, there are two possible categories of causes: physical causes, placed on the left side, and (control) software related causes, on the right.

Asking the question "why?" repeatedly, leads through the possible causes from top to down, eventually terminating in the source (cause), which are illustrated as the leafs of the tree in this case. There can be one or more such source causes in a diagram. If read from bottom to top, one can say that the cause on the other end is it's consequence. The causes of a standstill are manifold and can either be physically induced (such as e.g. a component failure, power failure, etc.) or software induced (e.g., bugs or attacks). In the former case, the causes can but do not necessarily have its origins in malicious behavior. In the latter case, it might be due to an attack of an intruder. It could be a direct attack, such as *denial of service* (DoS), or indirect, e.g. through malware. Based on this figure, it can be interpreted that in many cases, production standstills can probably be mitigated by following the seven touch points of software security as best practice security standard.

Consequences: The consequences of each of the causes can be seen one layer above the according layer, following the arrows in the opposite direction. Within the circle in the middle, there is the TD product which is the ultimate consequence of the causes.

TD Product: Loss of Know-how. Loss of know how takes place as soon as confidential information leaves the boundaries of the company in an uncontrolled way. For example, know-how is "leaked" if a planning document is sent to an external company or institution and the document contains more confidential information than necessary (E.g. specification data of components which do not affect the external company or institution). The worst case occurs if confidential information becomes known to competitors or other external parties who may profit from its abuse. The according cause and effect diagram is shown in Figure 3. The causes and the consequences are listed in the diagram the same way they are listed in the previous diagram (Figure 2) and described in paragraph IV.

Apart from the causes derived from the seven touch points, there is also the cause of an improper intellectual property (IP) protection, meaning that a customer could just rebuild the product of the company. Examples for confidentiality policy violations are e.g. to use a private email account to send confidential company data, or to discuss classified topics with company guests. Causes for the violation of the security policy can, but not necessarily have to be the absence of security operations that ensure that hardware and software measures exist, which hinder employees from such actions.

TD Product: Manipulation of Final Product. Manipulation of a product means any unauthorized changes in construction files that lead to unauthorized changes in the final product. Competitors could for example introduce slight changes in order to weaken a component and to reduce the overall lifetime or reliability of the final product. Such manipulations of the final product can influence product safety, lifetime or functions and thus affect the overall reputation of the company.

Figure 4 shows the diagram for the case of a final product which's specification has been manipulated due to the exploitation of a vulnerability. In the same way as in Figure 2), it shall illustrate how practical issues can in many of the cases be broken down to a deviation from the seven touch points.

V. Related Work

The term *Technical Debt* (TD) has its origins back in 1992, when Cunningham was the first to describe the paradox as an analogy to financial debt in order to help explain the phenomenon when "shipping the first time code" (an analogon to acquiring financial debt) in order to save time or effort. Paying interest is described by the author as every minute that is later on spent on this "not-quite-right" software code.

Since then, a lot of research has been conducted in the field - mainly focusing on the software engineering domain.

For example, Kruchten, Nord and Ozkaya describe TD within the software context and state that even a decision that is totally correct today can become TD in the future, When it later on becomes clear that in a retrospective, it would have been better [11]. Another example is the work of Letouzey, who presents SQALE, a method of TD evaluation within the software domain [18].

Vogel-Heuser et al. were among the first to describe the TD phenomenon within the field of automated production systems (APS) [12]. They point out that the development process of automated production systems is a lot more complex since more disciplines are involved at the same time (such as mechanical, electrical and control software engineering, in some cases also chemistry, etc.), that the process typically involves some extra phases, such as assembly and the startup that takes place onsite, and that testing during the development process is often not possible and can first be done after commissioning at the customer site. They developed SWMAT4aPS [19], a process to find TD within automated production systems, which they refined later on [15] and called TD4aPS. There, they select two case studies, a machine manufacturer and a plant manufacturer and show the cross disciplinary and cross lifecycle phase as two typical characteristics of technical debt in the APS domain.

In [20], Biffl et al. describe TD in CPS description languages, identify TD items within this field, and elaborate causes and effects of TD possibly arising. In [13], Biffl et al. report experiences with technical debt from a steel moulding company and create a cause and effect model to describe the relations and interplay of TD items and TD products. They refer to VDI 3695 [21] as gold standard. In [14], Biffl et al.elaborate on the relationship of TD items, TD causes and TD effects and model TD concepts as foundation for TD analysis and TD risk management. They investigate TD items, causes and their effects in the production system engineering process, regarding documentation and configuration management and found out how software engineers could benefit from product- and production knowledge modeling as a foundation for better understanding of the rationale of engineering design decisions.

The seven touchpoints of software security are a best practice standard introduced in 2004 by McGraw. Its goal is to show how security can be integrated in every phase of the creation of software with the aim to create systems that

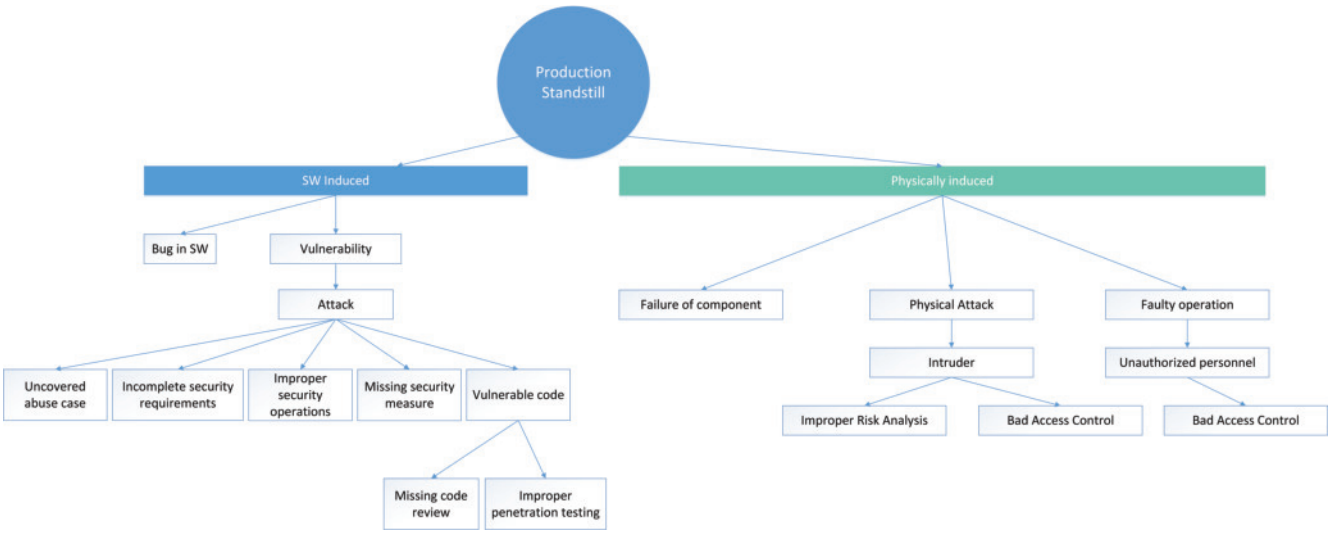

Fig. 2. Cause-Effect Diagram for the case of a production standstill.

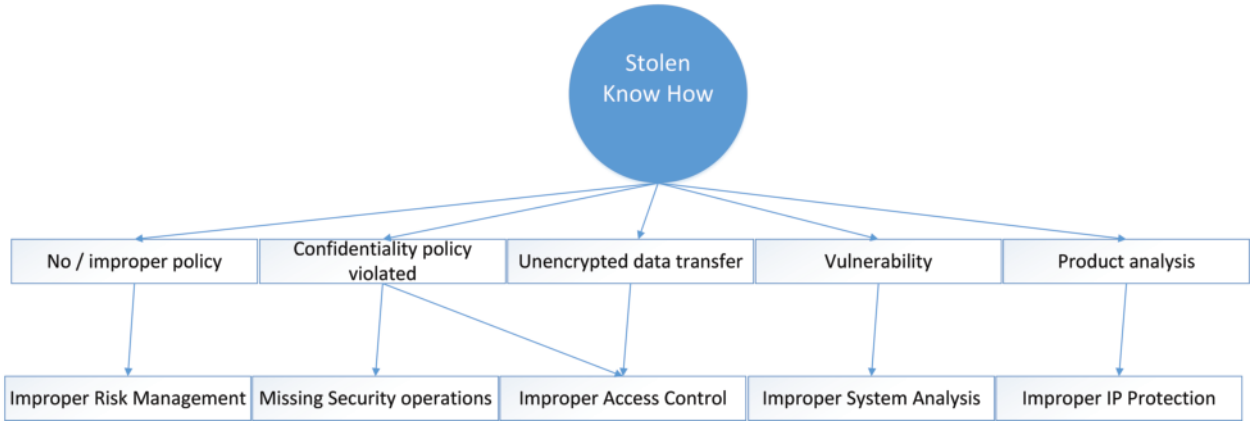

Fig. 3. Cause-Effect diagram for the case of stolen know-how.

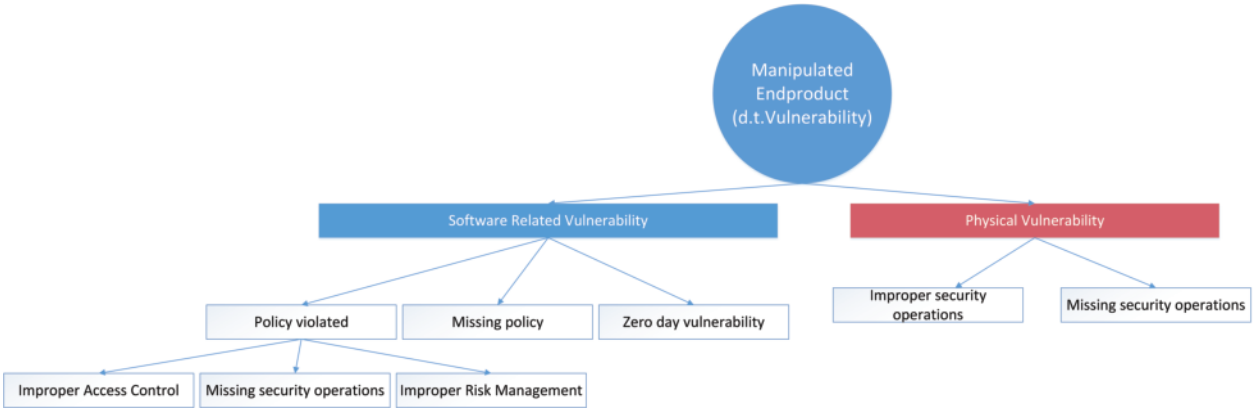

Fig. 4. Cause-Effect Diagram for the case of an unauthorized manipulation.

are "secure-by-design" as opposed to "adding security in the aftermath".

McGraw stresses that security by design is especially important when systems become more complex as in this case, adding security in hindsight becomes hardly possible.

VI. Conclusion and Future Work

Within the course of this paper, we have shown the differences and the similarities of software- and CPS security and that there are many similarities. We illustrated a possible way to analyze important symptoms, possible TD items and causes and how to map the seven touchpoints of software security, a set of software security best practices, in order to be used as gold standard to build cyber physical production systems that are secure-by-design, meaning that known attacks are countermeasured already during requirements, planning and construction of the CPS. This work is our first step in exploring the relationship between not following the seven touchpoints during the lifecycle of a CP(P)S and a concrete increase of costs due to TD as consequence. Therefore, a systematic framework to find concrete causes and their cross-relations, an exhaustive listing of possible cases of TD and their possible causes, and an evaluation of the former at several production companies shall follow. The next steps are the evaluation by measuring the rates of occurrence of these causes in practice, depending on specialization of the company, as well as the arising costs of the according TD items, in order to provide a reliable concept to the industry.

ACKNOWLEDGMENT

This material is based upon work partially supported by (1) the Christian-Doppler-Laboratory for Security and Quality Improvement in the Production System Lifecycle; The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the Nation Foundation for Research, Technology and Development is gratefully acknowledged; (2) SBA Research; the competence center SBA Research (SBA-K1) is funded within the framework of COMET Competence Centers for Excellent Technologies by BMVIT, BMDW, and the federal state of Vienna, managed by the FFG;

References

[1] ISA, "How industry 4.0 and digitization improves manufacturing responsiveness, quality and efficiency." [Online]. Available: https://automation.isa.org/industry-40-digitization-.../

[2] R. Von Solms and J. Van Niekerk, "From information security to cyber security," *computers & security*, vol. 38, pp. 97–102, 2013.

[3] M. Abomhara *et al.*, "Cyber security and the internet of things: vulnerabilities, threats, intruders and attacks," *Journal of Cyber Security and Mobility*, vol. 4, no. 1, pp. 65–88, 2015.

[4] G. McGraw, "Software security," *IEEE Security & Privacy*, vol. 2, no. 2, pp. 80–83, 2004.

[5] M. Howard, "Building more secure software with improved development processes," *IEEE Security & Privacy*, vol. 2, no. 6, pp. 63–65, 2004.

[6] B. Arkin, S. Stender, and G. McGraw, "Software penetration testing," *IEEE Security & Privacy*, vol. 3, no. 1, pp. 84–87, 2005.

[7] S. Gary McGraw, "Seven touchpoints for software security," accessed: 10-04-2019. [Online]. Available: http://www.swsec.com/resources/touchpoints/

[8] G. McGraw, "About the Author," accessed: 10-04-2019. [Online]. Available: http://www.swsec.com/about/

[9] W. Cunningham, "The wycash portfolio management system, addendum to the proceedings on object-oriented programming systems, languages, and applications (addendum)," 1992.

[10] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing technical debt in software engineering (dagstuhl seminar 16162)," in *Dagstuhl Reports*, vol. 6, no. 4. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[11] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *Ieee software*, vol. 29, no. 6, pp. 18–21, 2012.

[12] B. Vogel-Heuser, S. Rösch, A. Martini, and M. Tichy, "Technical debt in automated production systems," in *7th International Workshop on Managing Technical Debt (MTD)*. IEEE, 2015, pp. 49–52.

[13] S. Biffl, A. Lüder, F. Rinker, L. Waltersdorfer, and D. Winkler, "Experiences with Technical Debt in Parallel Multi-Disciplinary Systems Engineering," *Proceedings of Euromicro Software Engineering and Advanced Applications, Special Session on Technical Debt (SEaTeD)*, 2019, (in press).

[14] S. Biffl, L. Kathrein, A. Lüder, K. Meixner, and D. Winkler, "Software Engineering Risks from Technical Debt in the Representation of Product/ion Knowledge," in *Proceedings of SEKE*, 2019, (in press).

[15] Q. H. Dong and B. Vogel-Heuser, "Cross-disciplinary and cross-lifecycle-phase technical debt in automated production systems: two industrial case studies and a survey," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 1192–1199, 2018.

[16] M. Eckhart and A. Ekelhart, *Security and Quality Improvement for Engineering Flexible Software-Intensive Systems*. Springer, 2019, vol. 1, ch. Digital Twins for Cyber-Physical Systems Security: State of the Art and Outlook, p. 30.

[17] S. P. Skorobogatov, "Semi-invasive attacks: a new approach to hardware security analysis," 2005.

[18] J.-L. Letouzey, "The sqale method for evaluating technical debt," in *2012 Third International Workshop on Managing Technical Debt (MTD)*. IEEE, 2012, pp. 31–36.

[19] B. Vogel-Heuser, F. Ocker, and E.-M. Neumann, "Maturity variations of plc-based control software within a company and among companies from the same industrial sector," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*. IEEE, 2018, pp. 283–290.

[20] S. Biffl, A. Lüder, F. Rinker, L. Waltersdorfer, and D. Winkler, "Quality Risks in the Data Exchange Process for Collaborative CPPS Engineering," in *IEEE 17th International Conference on Industrial Informatics (INDIN)*, 2019.

[21] V. Guideline, "3695 Part 3: engineering of industrial plants - evaluation and optimization - subject methods," 2010.