Informatik Spektrum (2023) 46:96–103
https://doi.org/10.1007/s00287-023-01533-z

HAUPTBEITRAG

Safe and secure system architectures for cyber-physical systems

Frank J. Furrer[1]

Accepted: 20 December 2022 / Published online: 3 April 2023

Abstract

Cyber-physical systems are at the core of our current civilization. Countless examples dominate our daily life and work, such as driverless cars that will soon master our roads, implanted medical devices that will improve many lives, and industrial control systems that control production and infrastructure. Because cyber-physical systems manipulate the real world, they constitute a danger for many applications. Therefore, their *safety* and *security* are essential properties of these indispensable systems. The long history of systems engineering has demonstrated that the system quality properties—such as safety and security—strongly depend on the underlying *system architecture*. Satisfactory system quality properties can only be ensured if the fundamental system architecture is sound! The development of dependable cyber-physical architectures in recent years suggests that *two* harmonical architectures are required: a *design-time architecture* and a *run-time architecture*. The design-time architecture defines and specifies all parts and relationships, assuring the required system quality properties. However, in today's complex systems, ensuring all quality properties in all operating conditions during design time will never be possible. Therefore, an additional line of defense against safety accidents and security incidents is indispensable: This must be provided by the run-time architecture. The run-time architecture primarily consists of a *protective shell* that monitors the run-time system during operation. It detects anomalies in system behavior, interface functioning, or data—often using artificial intelligence algorithms—and takes autonomous mitigation measures, thus attempting to prevent imminent safety accidents or security incidents before they occur. This paper's core is the *protective shell* as a run-time protection mechanism for cyber-physical systems. The paper has the form of an introductory *tutorial* and includes focused references.

Context

Introduction

Cyber-physical systems are computer-controlled, networked systems that interact with the physical environment, often in a control loop, some of them in an autonomous way [1–6]. Typical examples include autonomous cars, autopilot in an airplane, a heart pacemaker, or cooperating robots in a manufacturing line. Because of their impact on the *real-world*, cyber-physical systems must be built so that they cannot harm or damage people, property, or the environment: Their behavior must be *safe* and *secure*. Engineering safe and secure cyber-physical systems has become a specific, exciting, and essential engineering discipline.

A long time ago, computers were just processing data, such as keeping accounts or managing inventory. Then they slowly started interacting with the physical world, for example, in the form of embedded computers controlling a combustion engine or as supervisory control and data acquisition (SCADA) systems governing industrial plants. Today, computers controlling all sorts of cyber-physical systems are pervasive—we find them everywhere. They have taken over control from small devices like a heart pacemaker to large applications, such as an autonomous container ship.

The system receives information about the environment from *sensors* (temperature, wheel rotation rate, camera, radar, gyroscope, etc.) and acts on the physical environment through *actuators* (motors, pumps, valves, etc.). The system comprises a number of interacting control algorithms, many of them closed-loop feedback algorithms. Some of these algorithms are based on self-learning (machine learning), for example, an autonomous vehicle's video processing software.

✉ Frank J. Furrer
frank.j.furrer@bluewin.ch

[1] Faculty of Computer Science, Technical University of Dresden, Dresden, Germany

Software

Cyber-physical systems are controlled by *software*, that is, most of their functionality is implemented in software. Control by software carries some *risks*: A failure, fault, error, or successful cyber-attack—either in the software or in the execution platform—can have grave consequences, such as safety accidents, security incidents, crashes, or casualties. In today's environment, malicious interactions, such as hacking, malware, infiltration, etc., can also inhibit the correct operation and lead to dangerous consequences. Therefore, the *quality properties* of the cyber-physical system—especially *safety* and *security*—must be assured during all phases of system development, operation, and evolution [7–12].

Architecture

At the heart of a cyber-physical system is its *architecture* [13–16]: "*Fundamental concepts or properties of an entity in its environment (= Context of surrounding things, conditions, or influences upon an entity) and governing principles for the realization and evolution of this entity and its related life cycle processes*" [17]. A long—and sometimes painful—history of systems has proven that adequate, sound architecture is indispensable [18]. The architecture provides the foundation for the efficient development and evolution of the cyber-physical system and enables to a large extent also the *quality properties*!

Safety and security

The list of a system's possible quality properties/attributes is extensive (e.g.: https://en.wikipedia.org/wiki/List_of_system_quality_attributes). For the cyber-physical system, the essential quality properties are *safety* (e.g.: [19]) and *security* (e.g.: [20, 21]).

Drift into failure

Fortunately, most modern *system engineering processes* are strongly safety and security aware [22–26]. In the majority of cases, these processes produce dependable and trustworthy systems. The organizations which make cyber-physical systems are almost always careful and diligent. Nonetheless, the press regularly reports security incidents and safety accidents. Why the discrepancy?

There are many reasons. First, the enormous *complexity* of today's (and even more: tomorrow's!) cyber-physical systems makes it impossible to avoid all vulnerabilities. Second, the operating environment of these systems becomes more hostile every year (higher probability of failures, greater sophistication of malicious activities). Third, the market pressure demands low development and production cost. Fourth, the high rate of change often entices the developers to "cut corners," that is, reduce or skip necessary quality assurance measures, such as modeling, reviews, verification, validation, and thorough testing. The result is an accumulation of *technical debt* [18, 26] and *architecture erosion* [18, 27]. This slow, hardly noticeable effect is called *drift into failure* [28] and constitutes a grave risk for evolving cyber-physical systems.

Last defense

As numerous examples show beyond doubt, it is not possible to eliminate all *vulnerabilities* from a complex cyber-physical system during development/extension/deployment time. Unfortunately, a likelihood always exists that the system will experience a security incident or generate a safety accident during operation.

Are there mechanisms other than a very diligent development process to reduce the impact/damage of a security incident or a safety accident? Fortunately, the answer is *yes* and reads: *Run-Time Monitoring* [29–34]. In run-time monitoring, the system's behavior is observed and automatically checked for compliance against the *desired* behavior. The desired behavior is defined in policies, specifications, rules, or models. The run-time monitor attempts to identify *anomalies*, that is, any deviation from the desired behavior. Preferably, the run-time monitor works in real-time: In this case, the monitor can detect, inhibit, or mitigate *anomalous behavior* before a safety accident or a security incident occur. The run-time monitor, therefore, acts as a *last line of defense* (Fig. 1): The system's engineering process attempts to eliminate the vulnerabilities in the system. However, a (hopefully small) number of vulnerabilities remain in the run-time system! A malicious threat or an unforeseen failure in the run-time system can thus provoke a security incident or generate a safety accident. If the run-time monitor works correctly and in real-time, it may *prevent*—or at least substantially *reduce* the negative impact—of the security incident or the safety accident. The functionality of the run-time monitor thus forms the last *line of defense* of the cyber-physical system!

Run-time monitoring and protective shell

Run-time monitoring principle

"Run-Time Monitoring as a Last Line of Defense" of a cyber-physical system is used increasingly in various industries (e.g.: [31, 35]). The principle of run-time monitoring is explained in Fig. 2: The real behavior is continuously *com-*

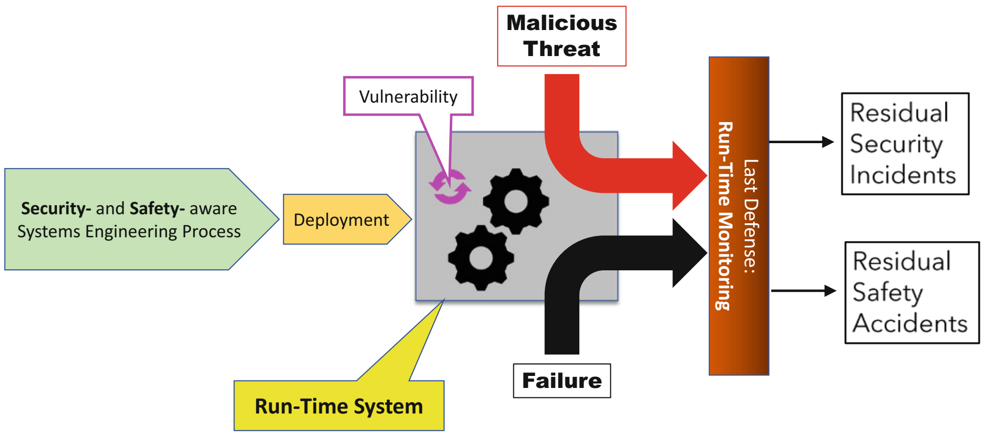
Malicious Threat
Vulnerability
Security- and Safety- aware Systems Engineering Process
Deployment
Run-Time System
Failure
Last Defense: Run-Time Monitoring
Residual Security Incidents
Residual Safety Accidents

Fig. 1 Run-time monitoring as last defense

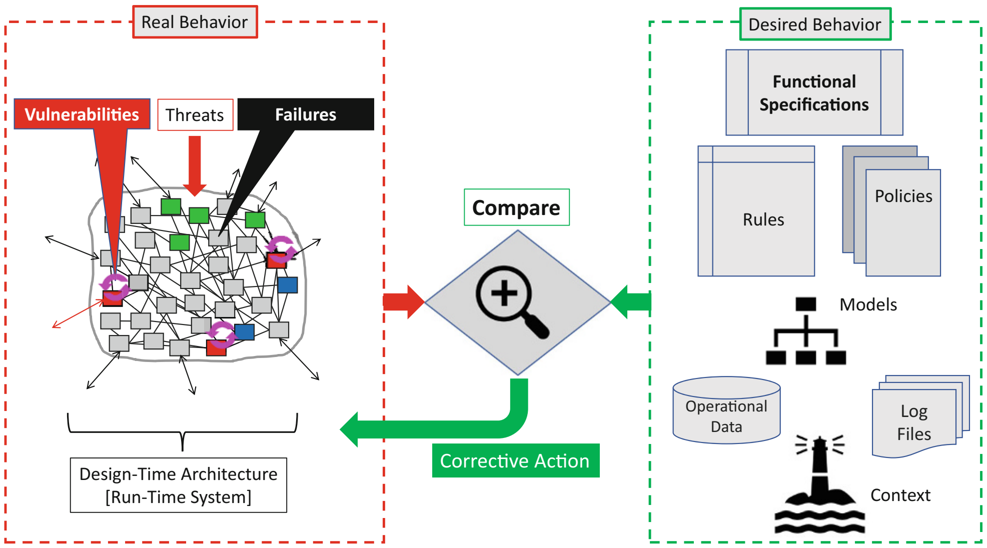

Fig. 2 Run-time monitoring principle

pared to the desired behavior. The desired behavior can be defined by a number of techniques:

- The *functional specifications*, expressed in a formal, machine-readable language (e.g.: [34, 40, 41]).
- A set of *policies*, expressed in a formal, machine-readable language [42].
- A set of *rules*, expressed in a formal, machine-readable language [32].
- Structural and behavioral *models*, expressed in a formal, machine-readable language [43, 44].
- In addition, the *comparison* makes use of information, such as operational data, log files, and the context (environment, partner systems, public information).

If a deviation of the real behavior from the desired behavior is detected, the run-time monitor takes *corrective action*, whenever possible in real-time. Many types of corrective actions are possible, all aiming to avoid or reduce the negative impact of a safety accident or security incident [12].

Using run-time monitoring (often called "active run-time monitoring" because of its real-time intervention capabilities) requires two types of *system architecture*:

1. The *design-time* architecture
2. The *run-time* architecture

Design-time architecture

The design-time architecture aims to *avoid* as many vulnerabilities in the system as possible. This is achieved by a diligent, security- and safety-aware system's engineering process and a subsequent vulnerability elimination process (Fig. 3: e.g.: [10]).

Run-time architecture

As soon as the design-time architecture of the cyber-physical system is judged to be sufficiently safe and secure, the system is deployed, that is, transferred to its operational environment and handed over to the users (Fig. 3). Unfor-

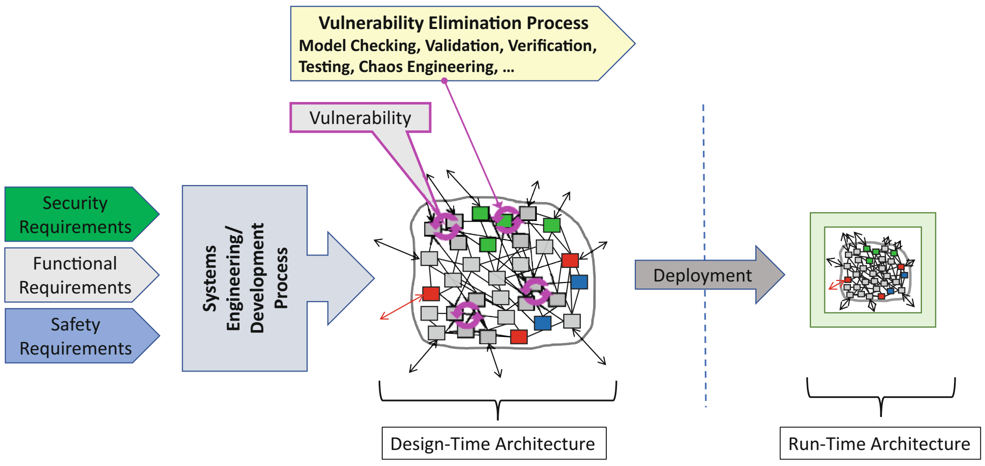
Vulnerability Elimination Process
Model Checking, Validation, Verification, Testing, Chaos Engineering, …
Vulnerability
Security Requirements
Functional Requirements
Safety Requirements
Systems Engineering/ Development Process
Deployment
Design-Time Architecture
Run-Time Architecture

Fig. 3 Design-time architecture

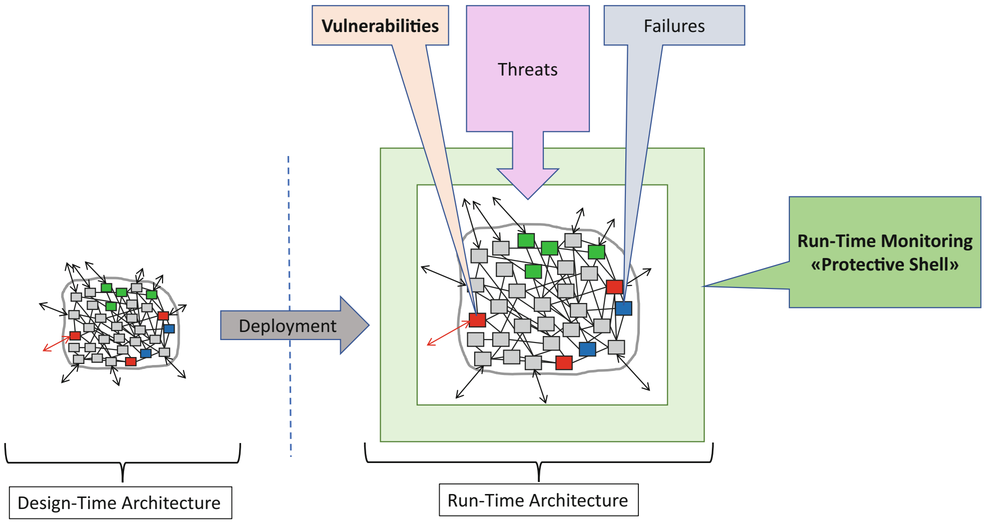

Fig. 4 Run-time architecture

tunately, the run-time system may still contain *vulnerabilities*—which constitute a considerable *risk* for its usage.

Therefore, an *additional* architectural element protects the run-time system: the (active) *run-time monitoring* (Fig. 4). The run-time monitoring embraces the run-time system and attempts to protect it from the impact and the consequences of threats and failures—whenever possible in real-time. This additional layer of protection can be seen as a *protective shell* that enfolds the running system. The idea of a protective shell as a *separate* architectural element and engineering artifact was presumably introduced by Lance Eliot under the name of "AI Guardian Angel Bots" for systems controlled by machine learning [36]. Here, the less exotic name *Protective Shell* is preferred [12].

Protective shell

The engineering design and the capabilities of a *protective shell* strongly depend on the run-time system to be protected. A *generic architecture* of a system with a protective shell is shown in Fig. 5. In the core of Fig. 5, the operational cyber-physical run-time system, including its interfaces to the real world and the network connections, is featured. Enfolding the run-time system is the protective shell. The protective shell disposes of *more* information than the run-time system from additional sources, possibly even from additional hardware. Examples of additional information sources include (Fig. 5):

- Operational data, log files, functional specifications, behavior models, policies, and specific rule sets
- Context information (From the environment, from other systems, from public sources, etc.)
- From access to the sensors (inputs) and actuators (outputs), possibly even using additional sensors or measuring instruments
- From the network usage, monitoring, and logging

In addition to traditional techniques, such as range and rate checks of sensors, and discrepancy and plausibility

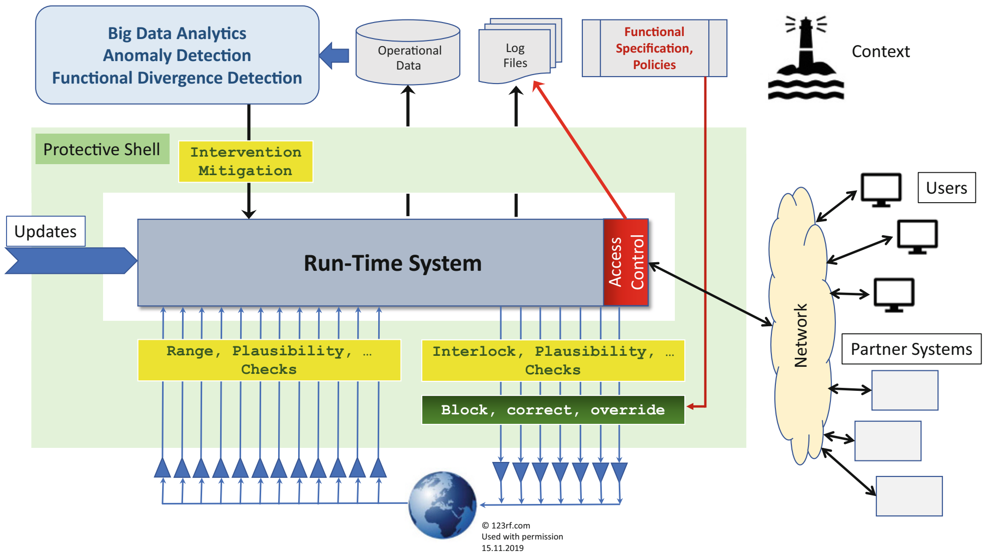

Fig. 5 Protective shell

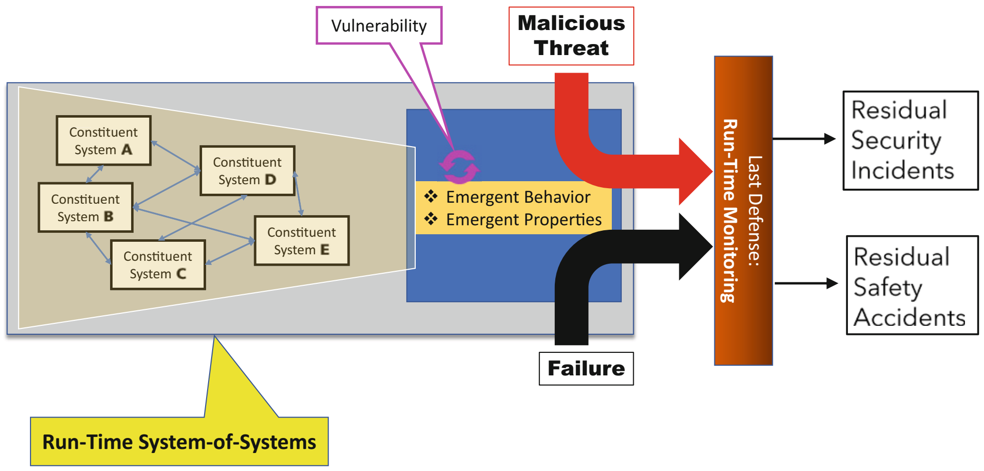

Fig. 6 Protection against emergent behavior

checks on actuator values, the protective shell often uses artificial intelligence and machine learning to detect *anomalies* [37–39, 45, 46, 50]. Any anomaly in behavior detected is immediately analyzed, assessed, and corrective actions are taken. Corrective action may include stopping the system, leading the system into a safe state, or into a safe degraded operation.

Emergent behavior

Most cyber-physical systems today consist not of one single, homogeneous system but are assembled from various constituent systems—thus forming a *system-of-systems* (Fig. 6; [51, 52]). A number of self-contained systems with specific functionality are interconnected to realize higher-level objectives. By combining the functionality of the constituent systems, superior functionality can be achieved, which cannot be provided by any of the constituent systems alone. An example is the various driver assistance systems in modern cars, such as lane-keeping, distance control, electronic stability control, traffic sign recognition, emergency braking capability, obstacle detection, automatic speed limiter, and airbags. Individually they offer assistance for specific potential accident situations. However, if the functionality of these systems is *combined*, a much safer car results. The emerging functionality from combining obstacle recognition with automatic emergency braking capability and electronic stability control will prevent significantly more accidents than each of the individual systems possibly could. This desired, valuable emergent functionality is the reason why the system-of-systems is designed and built!

Unfortunately, assembling system-of-systems from their constituent systems can also generate *unexpected, undesired, potentially damaging behavior*. The constituent systems' interconnection may generate unexpected failure modes, unanticipated system weaknesses, or new attack

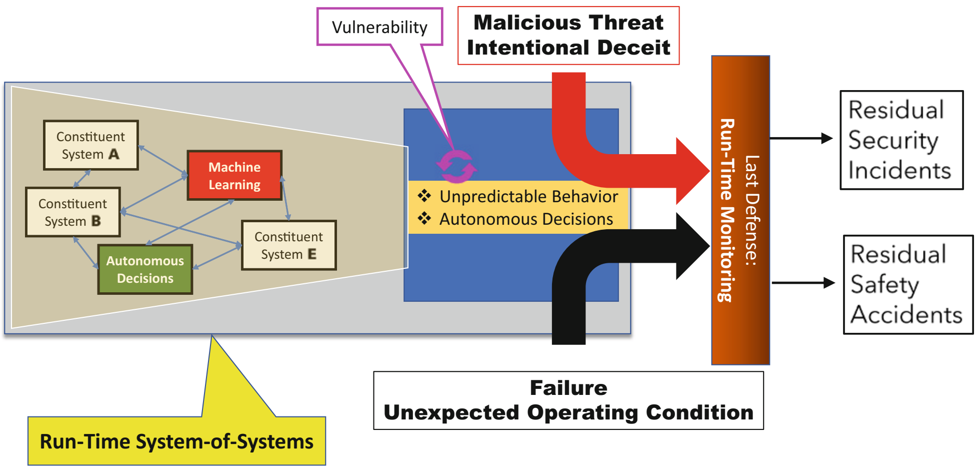

Fig. 7 Autonomy and machine learning

avenues—as negative, unintentional emergence [53, 54]! It could be suggested that a *protective shell* is the only defense against unexpected, dangerous emergent behavior.

Autonomy and machine learning

Modern cyber-physical systems exhibit a strong tendency towards *autonomous behavior* (e.g.: [55]): Such systems can change their behavior due to learning from experience or in response to unanticipated situations during operation. They are characterized by computers (i.e., software) making *decisions* affecting the physical world, such as autonomous vehicles. In many applications, these decisions are based on *machine-learning algorithms* [56–58], such as recognizing obstacles, their trajectories, and speeds from video, radar, or lidar images. Often, the machine-learning algorithms are not based on deterministic calculations but, for example, on statistical or training data evaluation. This can introduce a high degree of uncertainty and unpredictability in the autonomous system [36, 56, 59], which, in turn, introduces the risk of safety accidents or security incidents. Again, *anomaly detection* during run-time would be the last defense because predicting, assessing, and mitigating all safety and security risks during the development/deployment process is improbable in the context of autonomy and machine learning (Fig. 7).

Conclusions

A *protective shell* is a technique that can significantly enhance the safety and security of cyber-physical systems at run-time. It is a current, active research area, and some industries producing mission-critical cyber-physical systems are already implementing it.

However, the challenges of implementing a protective shell are that:

- Using a protective shell requires a very high degree of *formalization* for reliable anomaly detection [47].
- Designing a protective shell to protect against damaging run-time behavior is a highly challenging engineering task.
- The protective shell consumes additional run-time resources (power, CPU, memory).
- Designing and implementing a protective shell needs *highly educated engineers* [48].
- The protective shell's code and data increase the system's complexity, which may generate additional *failure modes* and possibly also enlarges the *attack surface* [49].

References

1. Alur R (2015) Principles of cyber-physical systems. MIT Press, Cambridge
2. Kravets AG, Bolshakov AA, Shcherbakov MV (eds) (2020) Cyber-physical systems—Industry 4.0 challenges. Springer Nature Switzerland, Cham
3. Möller DPF (2016) Guide to computing fundamentals in cyber-physical systems—Concepts, design methods, and applications. Springer, Cham
4. Rawat DB, Rodriques JJPC, Stojmenovic I (eds) (2016) Cyber-physical systems—From theory to practice. CRC Press, Boca Raton

5. Romanosky A, Ishikawa F (eds) (2017) Trustworthy cyber-physical systems engineering. CRC Press, Boca Raton
6. Song H, Rawat DB, Jeschke S, Brecher C (eds) (2016) Cyber-physical systems—Foundations, principles, and applications. Academic Press, London
7. Boulanger J-L (2013) Safety management of software-based equipment. John Wiley & Sons Inc, Hoboken
8. Dowd M, McDonald J, Schuh J (2006) The Art of software security assessment—Identifying and preventing software vulnerabilities. Addison-Wesley, Upper Saddle River
9. Hobbs C (2020) Embedded software development for safety-critical systems, 2nd edn. CRC Press, Boca Raton
10. Knight J (2012) Fundamentals of dependable computing for software engineers. CRC Press, Boca Raton
11. McGraw GR (2006) Software security—Building security. Addison Wesley, Upper Saddle River
12. Furrer FJ (2022) Safety and security of cyber-physical systems. Springer Vieweg, Wiesbaden
13. Nakajima S, Talpin J-P, Toyoshim M, Yu H (eds) (2017) Cyber-physical system design from an architecture analysis viewpoint. Springer Nature, Singapore
14. Liu C, Zhang Y (eds) (2019) Cyber-physical systems: Architectures, protocols, and applications. CRC Press, Boca Raton
15. Bass L, Clements P, Katzman R (2013) Software architecture in practice, 3rd edn. Pearson education, SEI-series. Addison-Wesley, Upper Saddle River
16. Rozanski N, Woods E (2012) Software systems architecture, 2nd edn. Addison-Wesley, Upper Saddle River
17. ISO/IEC/IEEE 42010:2022 (2022) Software, systems, and enterprise—Architecture description. https://www.iso.org/standard/74393.html. Accessed 10 Dec 2022
18. Furrer FJ (2019) Future-proof software-systems—A sustainable evolution strategy. Springer Vieweg, Wiesbaden. ISBN 978-3-658-19937-1.
19. Bahr NJ (2017) System safety engineering and risk assessment: a practical approach, 2nd edn. CRC Press, Boca Raton
20. Song H, Fink GA, Jeschke S (2018) Security and privacy in cyber-physical systems: foundations, principles, and applications. Wiley-IEEE, Hoboken
21. Brooks CJ, Craig PA Jr. (2022) Practical industrial cybersecurity: ICS, industry 4.0, and IIoT. John Wiley & Sons Inc, Hoboken
22. Axelrod CW (2012) Engineering safe and secure software systems. Artech House, Norwood
23. Deogun D, Bergh Johnsson D, Sawano D (2019) Secure by design. Manning, Shelter Island
24. Ackerman P (2021) Industrial cybersecurity: Efficiently monitor the cybersecurity posture of your ICS environment, 2nd edn. Packt, Birmingham
25. Hobbs C (2020) Embedded software development for safety-critical systems. CRC Press, Boca Raton
26. Kruchten P, Nord R, Ozkaya I (2019) Managing technical debt: reducing friction in software development. Pearson Education, London
27. Fairbanks G (2010) Just enough software architecture: a risk-driven approach. Marshall & Brainerd, Boulder
28. Dekker S (2011) Drift into failure: from hunting broken components to understanding complex systems. CRC Press, Boca Raton
29. Bartocci E, Falcone Y (eds) (2019) Lectures on runtime verification—Introductory and advanced topics. Springer Nature, Cham
30. Drusinsky D (2006) Modeling and verification using UML statecharts—A working guide to reactive system design, run-time monitoring, and execution-based model checking. Newnes, Burlington
31. Harrison L (2020) How to use run-time monitoring for automotive functional safety. https://www.techdesignforums.com/practice/technique/how-to-use-runtime-monitoring-for-automotive-functional-safety. Accessed 14 Dec 2022 (TechDesignForum White Paper)
32. Bhardwaj Haupt N, Liggesmeyer P (2019) A runtime safety monitoring approach for adaptable autonomous systems. In: Romanovsky A, Troubitsyna E, Gashi I, Schoitsch E, Bitsch F (eds) Computer safety, reliability, and security. SAFECOMP 2019. Lecture notes in computer science, vol 11699. Springer, Cham
33. Kane A (2015) Run-time monitoring for safety-critical embedded systems. Ph.D.Thesis, Carnegie Mellon University, Pittsburgh. https://users.ece.cmu.edu/. Accessed 14 Dec 2022
34. Taimoor Khan M, Serpanos D, Shrobe H (2016) Sound and complete run-time security monitor for application software. Preprint, arXiv:1601.04263v1 [cs.CR]. https://www.researchgate.net/publication/291229626_Sound_and_Complete_Runtime_Security_Monitor_for_Application_Software. Accessed 12 Dec 2021
35. Janicke H, Nicholson A, Webber S, Cau A (2015) Run-time-monitoring for industrial control systems. Electronics 4(4):995–1017. https://doi.org/10.3390/electronics4040995
36. Eliot L (2016) AI guardian angel bots for deep AI trustworthiness: practical advances in artificial intelligence (AI) and machine learning. LBE
37. Weber H (2019) Big data: a complete guide to the basic concepts in data science, cyber security, analytics and metrics
38. Prabhu CSR, Chivukula AS, Mogadala A, Ghosh R, Livingston LMJ (2019) Big data analytics: systems, algorithms, applications. Springer Nature, Singapore
39. Li M (2022) Multi-fractal traffic and anomaly detection in computer communications. CRC Press, Boca Raton
40. O'Regan G (2017) Concise guide to formal methods: theory, fundamentals, and industry applications. Springer, Cham
41. Bartocci E, Deshmukh J, Donzé A, Fainekos G, Maler O, Ničković D (2018) Specification-based monitoring of cyber-physical systems—A survey on theory, tools, and applications. In: Bartocci E, Falcone Y (eds) Lectures on runtime verification. Lecture notes in computer science, vol 10457. Springer, Cham
42. ETH (Eidgenössische Technische Hochschule) Information Security Group (2022) Runtime policy monitoring and enforcement. https://infsec.ethz.ch/research/projects/mon_enf.html. Accessed 17 Dec 2022
43. Höfig E (2011) Interpretation of behaviour models at runtime: performance benchmark and case studies. Südwestdeutscher Verlag für Hochschulschriften, Saarbrücken
44. Ardagna D, Zhang L (2010) Run-time models for self-managing systems and applications (autonomic systems). Birkhäuser, Basel
45. Dunning T, Friedman E (2014) Practical machine learning: a new look at anomaly detection. O'Reilly Media, Sebastopol
46. McCarthy J, Powell M, Stouffer K, Tang CYT, Zimmerman T, Barker W, Ogunyale T, Wynne D, Wiltberger J (2020) Securing manufacturing industrial control systems: behavioral anomaly detection. https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8219.pdf. Accessed 17 Dec 2022
47. Wang J (2020) Formal methods in computer science. Routledge, Boca Raton
48. U.S. National Academy of Engineering (2004) The engineer of 2020: Visions of engineering in the new century. https://nap.nationalacademies.org/download/10999. Accessed 28 Dec 2022
49. Hollnagel E (2018) Safety-II in practice: Developing the resilience potentials. Routledge, Milton Park
50. Ninagawa C (2023) AI time series control system modelling. Springer Nature, Singapore
51. Jamshidi M (ed) (2009) Systems of systems engineering—Innovations for the 21st century. John Wiley & Sons Inc, Hoboken
52. Luzeaux D, Ruault J-R, Wipplere J-L (eds) (2011) Complex Systems and Systems of Systems Engineering. iSTE, London

53. Rainey LB, Jamshidi M (eds) (2019) Engineering emergence—A modeling and simulation approach. CRC Press, Boca Raton
54. Mittal S, Diallo S, Tolk A (eds) (2018) Emergent behaviour in complex systems—A modeling and simulation approach. John Wiley & Sons, Hoboken
55. Ivancevic VG, Darryn JR, Pilling MJ (2017) Mathematics of autonomy—Mathematical methods for cyber-physical-cognitive systems. World Scientific, Singapore
56. Lawless WF, Ranjeev Mittu R, Sofge D, Russell S (eds) (2017) Autonomy and artificial intelligence—A threat or savior? Springer, Cham
57. Dunning T, Friedman E (2014) Practical machine learning: a new look at anomaly detection. O'Reilly Media, Sebastopol
58. Burkov A (2020) Machine learning engineering. True Positive, Quebec City
59. Faulkner A, Nicholson M (2020) The emergence of accidental autonomy. In: Parsons M, Nicholson M (eds) Assuring Safe Autonomy. Proceedings of the 28th Safety-Critical Systems Symposium (SSS'20), York, UK, 11–13 February 2020