DOI: 10.15439/2022F287
ISSN 2300-5963 ACSIS, Vol. 31

Integrated Checklist for Architecture Design of Critical Software Systems

Adela Bierska, Barbora Buhnova and Hind Bangui
Faculty of Informatics, Masaryk University
Brno, Czech Republic
{bierska, buhnova, hind.bangui}@mail.muni.cz

***Abstract*—With the advancement of digitalization, critical information infrastructures, such as intelligent energy distribution, transportation, or healthcare, have opened themselves towards intelligent technological opportunities, including automation of previously manual decision making. As a side effect, the digitalization of these infrastructures gives rise to new challenges, especially linked to the complexity of architecture design of these infrastructures, to later support necessary software quality and safeguard the systems against attacks and other harm. To support software architects in the design of these critical software systems, well structure architectural knowledge would be of great help to prevent the architects from missing some of the crucial concerns that need to be reflected with built-in architectural mechanisms, early during architecture design.**

Given the narrow scope of existing guidelines, with the need of browsing and combining multiple sources, this paper proposes an integrated checklist to cover the breath of architectural concerns for the design of critical software systems, covering the need for built-in mechanisms to prevent, detect, stop, recover from and analyse intentional as well as unintentional threats to system dependability. Contrary to existing guidelines that typically focus on runtime incident handling, our checklist is to be used during architecture design to ensure that the system has built-in mechanisms to either handle the incidents automatically or include the right mechanisms to support the runtime incident handling.

***Index Terms*—Software architecture, design checklist, critical information infrastructure, dependability**

I. Introduction

CRITICAL information infrastructures could be understood as digital and vital systems that require immediate attention and protection in modern cities (e.g., intelligent transportation) because they contribute in the improvement of the quality of life and sustainable development of our society. Over the past decades, critical infrastructures in various domains of human life have become largely digitized, stressing achievement of system security, resilience, reliability and other characteristics of failure-free and dependable operation [1], [2]. However, although these systems have become highly software intensive, software architecture experts have often not been involved in the design of these systems, which is why the operators of these systems are seeking software architecture expertise ex-post to evaluate and improve software architecture design of these systems.

This research was supported by ERDF "CyberSecurity, CyberCrime and Critical Information Infrastructures Center of Excellence" (No. CZ.02.1.01/0.0/0.0/16_019/0000822).

While software architects have access to numerous standards and guidelines for system design and auditing in concrete domains of critical infrastructures, e.g., CIPSEC [3] or Cybersecurity Certification [4], there is no general overview of design guidelines they shall consider, which is leaving them with substantial risk that they might miss some crucial consideration. More so that existing standards and guidelines disproportionately more focus on hardware considerations, which might make it even more likely for software architect that they miss some software-architecture related aspects.

To address this gap, the aim of this paper is to propose the creation of an integrated checklist supporting software architects in the design of critical software systems. The checklist is meant to cover guidelines that help the architect to design buit-in mechanisms to improve the dependability of the designed system. Given the existence of various low-level tactics and patterns, e.g. for availability, reliability or security in specific types of systems [5], [6], the suggested critical infrastructure design guidance shall be high-level and integrative, emphasizing the specifics of critical information infrastructures that might otherwise be missed. This paper introduces the reader to the context of critical infrastructures and software incidents and explains the checklist creation process and its usage, with additional supplementary material available for download at [7].

The structure of the paper is as follows. After the introduction, the context of the topic together with the state of the art and related work is presented in Section II. Section III lays down the design considerations guiding the design of the checklist. Section IV details the methodology used to create the checklist, after which the resulting checklist is presented in Section V. Additionally, supplementary material is available at [7], containing the full guidelines classification data, detailed guidelines descriptions, and a demonstration of the checklist in a real-life context.

II. Critical Systems and Infrastructures

There are numerous definitions of the term *critical (information) infrastructure (CI)*[1] in the literature from legal, political, technical, economical, geographical or social perspectives [8]. The German Federal Ministry of the Interior,

[1]The word *information* within *critical information infrastructures* is being used to emphasize reference to ICT enhanced critical infrastructures.

©2022, PTI

for instance, defines critical infrastructures as: "organizational and physical structures and facilities of such vital importance to a nation's society and economy that their failure or degradation would result in sustained supply shortages, significant disruption of public safety and security, or other dramatic consequences" [9]. Overall, there is an agreement that critical infrastructure is an infrastructure that is needed to keep other major technical and/or social systems running or which is needed to provide goods or services that are considered vital to the functioning of modern society [8].

1) Challenges in Architecture Design of CIs: Quality engineering is an inherent goal of software architecture design in any domain, driving the main software architecture activities, such as encapsulation, partitioning, or legacy components integration, using techniques such as isolation, redundancy allocation, or distribution [5], [10].

While in more wide-spread and popular domains, such as enterprise systems and web applications, many architectural patterns and styles exist [11], [12], integrated guidance for critical infrastructures is so far missing.

Obviously, various low level recommendations exist for specific infrastructures and quality attributes to be optimized [5], [6], where the infrastructures differ according to the local context of each nation, and the quality attributes differ according to the infrastructure, with many specific attributes not covered in existing software architecture literature (e.g. absorbtion as the ability to absorb the effects of system failures and thus minimize the consequences, or preparedness as the ability to withstand the expected crisis situations). This guidance, besides being isolated and localised, is moreover scarce and often provided in terms of *answers* rather than *questions to be asked*, which makes the critical infrastructure design process highly error-prone.

2) SW Architecture Design Checklist: Simple checklists can help reduce human error dramatically. As emphasized by Ivar Jacobson when advocating for software project checklists [13], "Neil Armstrong had a checklist printed on the back of his glove to ensure he remembered the important things as he made history as the first person to walk on the moon, so why not utilize them to keep software projects on track." Checklists in software engineering are not a new concept, being employed to facilitate software development processes by efficiently guiding industry experts and professional developers.

III. Design Considerations for Dependable CIs

Critical infrastructures are by definition safety-critical and often handle sensitive, secret, or in other way valuable data [14]. Consequently, they are an attractive target for attackers and need more robust protection.

Software-intensive critical infrastructures are highly complex as they operate with various technologies and have to be highly reliable. While designing the system, an architect may forget numerous basic features and cause vulnerabilities. This could endanger people or cost money. Our checklist is designed for these infrastructures to help its architects to meet the crucial safety and security standards and meet their high reliability expectations.

In this section, we set the foundations for the checklist, discussing its scope and analysing the types of incidents it shall cover, supporting the software architecture mechanisms to prevent, detect and handle them.

A. Domains

The most common CIs around the world, according to CIPSEC [15] are in the domains of health, energy, transportation, finance, food, water, and civil administration. Each of these CIs faces a different set of threats and safety concerns, but in their essence, the infrastructures need to meet very high standards that are very similar among them. In the presented version of the checklist, we thus focus on the design concerns that need to be reflected by all of them.

CI domains are usually interdependent [16]. For example, all of those mentioned above depend on the energy sector as they use software systems and machines for their operations. The energy sector relies on water supply used for cooling, which is impossible without transporting material within the infrastructure [15]. This can cause a domino effect throughout the whole system [17]. That is why it is necessary to stop an error as soon as possible and ensure the functioning of the most critical parts.

B. Types of Incidents

A software incident is an event that brings the system to an unwanted, anomalous state [18]. Incidents in CIs can be catastrophic and endanger human lives or the economy. Therefore software should be prepared to prevent and stop them.

We can classify incident causes in different ways. They can be intentional or unintentional, man-made or caused by natural disasters, caused by an error in code or hardware [19].

One of the most significant challenges of software dependability is, for the time being, unknown threats and attacks. New types of vulnerabilities are discovered every day [20], and natural disasters are often also unpredictable. Due to this, the software architect may not be aware of perils that will be ordinary for the system in a few years, months, or even days. Therefore we should prepare the system universally and not rely on a concrete list of possible threats.

1) Natural-Disaster Causes: Natural incidents are caused by natural disasters like floods, tornados, earthquakes, etc. [21] They can cause damage to the system hardware and therefore impact the correct functioning of the software. From the software point of view, we can also include in this category the incidents triggered by hardware errors (i. e., outage of parts) or by other damage to the hardware (i. e., incompetent manipulation or militarized attack). Natural disasters can be unpredictable, devastating, and cause a domino effect on other domains of CIs [21]. This enormously complicates the possibilities of testing the system's behavior during these incidents [22].

2) Man-Made Causes: By man-made incidents, we mean inadvertent mistakes made by a user or a premeditated attack from an attacker. In contrast to natural incidents, we can prevent many of these.

a) Accidental Errors: There are many reasons why users can make mistakes while using the system. For example, they may not know how to use the system, they can be tired and careless, or the system may be confusing [23]. In any case, this inappropriate usage should not endanger the correct functioning of the system.

b) Deliberate Attacks: Deliberate attacks on CIs occur increasingly often [24]. Main objectives of attackers are [25]:

- **Corruption of information:** Attackers try to change or damage data stored in the system or corrupt communication.
- **Denial of service:** Attackers try to overload or disrupt the system to become unavailable for authorized users.
- **Disclosure of information:** Attackers try to obtain private data or publish them to unauthorized entities.
- **Theft of resources:** Attackers try to access and misuse the system resources or provide them to unauthorized entities.
- **Physical destruction:** Attackers try to cause physical damage using the system.

The software should be prepared for all kinds of attacks, prevent them, and ensure safety in case of malicious usage. In contrast to accidental errors, deliberate attacks may last longer and be more complex. Attackers systematically conceal their activity, so detecting such incidents can be tricky.

C. The Role of Checklists

A checklist is a structured list of requirements or steps needed to achieve the given goal. It can have various structures [26] but should be brief and synoptical to minimalize the cognitive load of its usage [27]. When designing a software system, we can come across checklists, for example, in chosen standards. Checklists usually contain a list of conditionals that need to be fulfilled to meet the given standard [26].

In this work, we were inspired by checklists used by experts to facilitate their decision-making, e.g., in healthcare. Here, it can help us on two different levels – to make our decision (or diagnosis) or to check the correctness of our already-made decision [27]. Experts often tend to evaluate the system based on experienced patterns, which, however, may be superficial. In this case, the checklist reminds them of essential points that they should consider [28].

IV. Methodology

To ensure the comprehensiveness and completeness of the checklist, we had to collect guideline sources to base the checklist on. First, we gathered 32 standards related to software or software-intensive CIs based on criteria in Section IV-A, filtered them to select the representatives with complete coverage of the others, prioritizing freely available sources so that the architect can be pointed to them from the checklist [29], [30], [31], [32], [33], [34], [35], [36]. We collected all requirements meeting our inclusion and exclusion criteria (see section IV-B) from these standards and extracted the most common categories (i.e., Authorization, Authentication, Data protection, Logging, Input and Output, Network, Safety Ensuring, Backups, Encryption, and Third-Party Components).

We added categories to cover incident phases described in Section IV-C, i.e., Access Control, Anomaly Detection, Phenomenon Evaluation, Stopping from Propagating, Self-Adaptiveness, and Evidence. For each category, we went through existing studies and other sources [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [57] to examine the completeness of the coverage by the standards and complement the missing pieces. All these recommendations were again validated against our inclusion and exclusion criteria.

Finally, we revised categories, some of them were merged or removed, but we also added some to facilitate orientation within the checklist.

A. Sources of Guidelines

The checklist is based on two categories of sources: standards and other recommendations. Each has slightly different conditions for inclusion, but both must be primarily software-related.

Standards must be official and published by an approved organization, government department or peer-reviewed publisher. They should contain a set of rules and aims which ensure software security or safety. They can be designed for critical infrastructure in general, but they have to include a part aimed concretely at software.

When choosing standards, we primarily focused on their domain subsumption to critical infrastructures. Some of them are universal to all software. In that case, they should mention critical systems in their scope or focus on technology commonly used in our target domains.

By recommendations, we mean the research studies and books that software architect can study and abide by to accomplish system security and reliability. It is out of the scope of the checklist to go in a fine detail, it shall rather focus on a general guidance with great coverage in terms of the breath, not depth.

B. Scope

To fully utilize the advantages of the checklist form (e.g., briefness, coverage), we have to set strict inclusion and exclusion criteria of guidelines. The checklist should be compact but cover all identified categories.

1) Inclusion Criteria: Every guideline should be generally usable within CI systems. There may be exceptions for particular systems, but in general, all guidelines should focus on the design of dependable (safe, reliable and secure) critical systems. All guidelines must be relevant to the software architect and propose design improvement to software systems. The included guidelines shall reflect that not only do we try to prevent problems or incidents, but we are also designing systems to be able to operate safely in presence of such problems and incidents, i.e. we have to prepare the system to withstand accidents that have penetrated. Even with the robust, firmly secure architecture, we cannot presume that incidents are impossible [58].

2) Exclusion Criteria: (a) Human Resources: Many cyber-attack related studies support our incident handling phases division but aim more at management and incident planning [59], [60], [61]. Human resources and management are also excluded from the checklist scope. *(b) Hardware:* Considering that our guidelines focus on software, we exclude all purely-hardware related items. Hardware solutions are often irrelevant for software designers and strongly depend on their domain and chosen technologies. *(c) Coding Practices:* The choice of the programming language [62], frameworks and libraries can significantly affect the complexity of the development process. However, that shall is the task of the software architect to set constraints on such low level of detail. Therefore purely coding practices are excluded from the scope.

C. Incident-Handling Phases

The process of incident handling, which needs to be supported by the architecture design checklist, can be structured in multiple phases. In this work, we use the phases inspired by the *Computer Security Incident Handling Guide* [18], which sets them for handling already running incidents. In our case instead, we employ the phases to structure the guidelines *to design software architectures towards preparedness for these phases* making systems more robust and dependable (reliable and secure). The phases are:

- **Prevention:** The prevention of incidents strongly depends on the overall environment of the system. To specify guidelines for this phase, we have to consider all potential sources of failure and use appropriate architectural guidelines to safeguard all the possible entry points for the incidents to enter the system, taking both external and insider attacks, as well as events such as natural disasters into consideration.
- **Detection:** The primary aim of this phase is to design the system with built-in mechanisms to detect a running incident. That is to detect anomalies in the behaviour and discover intruders who might steal data without changing system behaviour. With the correct Detection in place, the system can switch on time to the Containment phase. The secondary (not less important) purpose of the Detection phase is to classify the fault and find its source. These are essential for stopping the fault from propagating and its following elimination.
- **Containment:** This phase covers tactics prepared for an immediate reaction to the detected incident, mainly stopping the problem from propagating and ensuring safety. While sometimes we need to stop the fault as soon as possible, in other cases we may only consider slowing down the attacker or using sandboxing. In any case, it is essential to bypass critical parts of software, especially those responsible for ensuring safety. Furthermore, the checklist shall motivate the architects to identify the must-work functionalities, on which the safety depends, and ensuring these parts work.
- **Recovery:** Right after getting the fault under control, the system shall have the right mechanisms to start the recovery process. It should be able to isolate and remove all infected and suspicious parts, as damaged system is more vulnerable to recurrent failures and attacks. Pre-incident preparation is fundamental as we need to compare states before and after and restore backup data.
- **Post-Incident Analysis:** An essential part of the post-incident analysis is forensic investigation, so the system should be designed and prepared for it before the incident occurs. Pieces of evidence that the system stores should be admissible at court of law and comply with local constraints to data monitoring and storing (e.g., GDPR).

The phases are not strictly separated, some guidelines can support multiple phases. For example, prevention must remain stable during the incident to impede collateral attack. Collection of evidence starts with recognizing the incident [63]. This classification of phases ensures its clear coverage of the security and reliability of the designed system. It also provides better possibilities for synoptical structure.

V. Integrated Checklist for Designing CIs

The primary purpose of checklists, in general, is to keep track of particular processes or units. They remind us of all steps we have to fulfill to complete the task and record every subtask we have already accomplished. If well designed, we can use them for self-evaluating without a checkup from another entity.

Our checklist provides this feature of self-evaluation of software systems by listing the must-haves for handling particular phases of security incidents and offering relevant standards and sources. After going through all of the provided guidelines, the software architect should be able to make sure that the architecture contains buit-in mechanisms to reflect all the given concerns.

All systems are unique and struggle with various issues. The checklist should cover the plentifullest amount of them and stay adaptive and concrete. Therefore it offers the available guidelines fulfilling our scope, in its breath, and also allows the architect to choose which are relevant.

Last but not least profitable feature of the checklist is facilitating communication in the development team. It should be readable by all team members independently on whether they are creators.

A. Structure

The structure has to observe typical qualities of checklists: briefness and clarity. Its main aim is to classify, sort external sources, and show only short descriptions and references. Any user should be able to browse all offered guidelines without previous knowledge of the checklist.

The form of the checklist is variable. Initially, the designer gets all the sources categorized by incident phases and tags.

TABLE I
PHASE 1: PREVENTION

GUIDELINE	TAGS	SOURCES
Data Protection		
Saved data are secured.	Data Protection, Encryption, Network	[32], [37], [38]
Network communication (internal too) is secured.	Data Protection, Network, Authentication	[32], [29], [33], [37], [36], [38]
Authorization		
Each entity has a specific role within the system with minimal necessary rights.	Authorization, Access Control	[32], [33], [39], [36]
Authorities can assign features to roles or concrete entities.	Authorization, Access Control	[40], [32], [29], [33], [36]
Each entity has access to the minimal amount of data possible.	Authorization, Access Control	[32], [29], [33], [38], [36]
There is an access timeout (or other control) set up and automatically disconnects the entity in case of inactivity.	Authorization, Access Control	[32], [33], [41], [36]
Authentication		
All (both local and remote) access should be protected by a unique entity identification and password, token, biometrics, or multi-factor authentication.	Access Control, Authentication	[32], [29], [33], [42], [37], [36]
Unsuccessful login attempts are logged and limited.	Access Control, Authentication, Logging	[32], [33], [36], [39], [44]
Used third-party technologies and components are certified that do not circumvent set IDs or passwords.	Access Control, Authentication, Third-party components	[32]
Used third-party configurations, updates, or other provided data use digital signatures.	Authentication, Third-party components	[32], [36]

Each guideline has one main phase and one main tag to be classified by. First-level classification is based on the incident phase, and is presented in Tables I–V. Inside these categories, guidelines are sorted by their main tag. This checklist view contains every guideline only once and is destined for the first walk-through to get to know all included recommendations. During incident planning, the architect should use it to make sure they reflect all the essential concerns and consider their properties. The phase-tags tree here may be a little more important than the recommendations themselves.

The architect should use the checklist either to guide the design process or at the end of the designing process to check that all the mentioned concerns are reflected. In fact, instead of ticking boxes, it is recommended to briefly *describe* the planned or realized extent to which the concern is reflected within the designed system. This can also improve communication within the development team or with stakeholders. Furthermore, when used to inspect the designed architecture, we recommend to annotate each concern with *strengths* and *weaknesses* of the designed solution with respect to the specific concern.

TABLE II
PHASE 2: DETECTION

GUIDELINE	TAGS	SOURCES
Logging		
All key events are logged.	Logging, Evidence	[46], [32], [47], [36]
Logs contain all important identification and classification.	Logging, Evidence	[32], [33], [36], [47]
Logs have various priority levels.	Logging	[32], [47]
There is a supervisory system that monitors logs and highlights alarms and anomalies.	Logging, Anomaly Detection	[32], [47], [48]
There is a mechanism that monitors if the logging system works.	Logging, Anomaly Detection	[32], [36]
Anomaly Detection		
The system recognizes distinct changes in configuration.	Anomaly Detection	[34], [29], [36]
The system recognizes unexpected or incomplete resets.	Anomaly Detection	[34]
The system recognizes memory failures.	Anomaly Detection	[34]
The system recognizes suspicious instructions.	Anomaly Detection	[34]
Anomalies in system performance are recognized and reacted to.	Anomaly Detection	[46], [48]
Anomalies in process behavior are recognized and reacted to.	Anomaly Detection	[46], [48]
File and directory changes are recognized and reacted to.	Anomaly Detection, Data Protection	[36], [46]
Input and Output		
User inputs and commands are validated and tested for sanity.	Input and Output	[49], [35]
External data are validated on entry.	Input and Output, Authentication, Third-party Components	[35]
The system controls all data before processing.	Input and Output	[34]
Phenomenon Evaluation		
Fault severity is classified into multiple levels.	Phenomenon Evaluation, Logging	[36], [48]
The system has a precisely defined failure tolerance threshold.	Phenomenon Evaluation	[32], [48]
Before launching a critical mode, the system checks if the trigger is valid.	Phenomenon Evaluation, Safety Ensuring	[32], [48]
Network		
Network data are collected.	Network, Evidence	[46]
All network alerts and error reports are checked.	Network, Anomaly Detection	[46]
The system recognizes unexpected, unusual, or suspicious traffic.	Network, Anomaly Detection	[46], [37]
Unauthorized entities connected to the system's network are recognized and restricted.	Network, Anomaly Detection, Authorization, Third-party Components	[29], [33], [36], [46]

TABLE III
PHASE 3: CONTAINMENT

GUIDELINE	TAGS	SOURCES
Stopping from Propagating		
The system is divided into independent parts with the possibility of a partial shutdown.	Stopping from Propagating	[34]
Safety-critical functions are isolated from non-safety-critical.	Stopping from Propagating	[46]
The system uses sandboxing to encapsulate high-risk parts.	Stopping from Propagating	[50]
Safety Ensuring		
The system is tolerant of an unstable or missing power source.	Safety Ensuring	[34]
Safety-critical software requirements are precisely identified and described.	Safety Ensuring	[30], [46]
There are *must-work functions* identified within the system.	Safety Ensuring, Self-Adaptiveness	[34], [51]
Must-work functions are redundant.	Safety Ensuring, Self-Adaptiveness	[34], [36], [51], [52]
Each of the *must-work functions* has at least two independent ways to control them.	Safety Ensuring	[34], [36]

TABLE IV
PHASE 4: RECOVERY

GUIDELINE	TAGS	SOURCES
Backups		
Critical data have a backup.	Backups	[29], [37]
Backups are off-site to be protected from local disasters (e.g., fire, flood, ...).	Backups	[36], [53]
Backup time intervals vary based on the frequency of changes.	Backups	[53]
Backups are protected from unauthorized access.	Backups, Access Control	[53]
Every backup process is checked to see if it was successful.	Backups	[29]
Before recovery from backup, data are preserved for further analysis of the incident.	Backups, Post-Incident Analysis	[29]
Self-Adaptiveness		
Must-work functions have a self-healing mechanism.	Self-Adaptiveness	[51]
System is prepared to adapt to operating without damaged parts.	Self-Adaptiveness, Safety Ensuring	[51]

B. Tags

Tags serve for a more detailed classification of the guidelines. Overall, they are based on system procedures, features, possibly used technologies, or specify parts of the incident phase more accurately. The tags, selected based on our methodology in Section IV, were also validated against the 20 common security requirements defined by CIPSEC [15]. They cover all these requirements except one requirement (number 15) that is management-oriented and thus out of the checklist scope, and add some requirements that were not covered by CIPSEC.

TABLE V
PHASE 5: POST-INCIDENT ANALYSIS

GUIDELINE	TAGS	SOURCES
Logging		
Logs are archived.	Logging, Evidence	[46]
Logs are secured.	Logging, Encryption	[46]
Old or useless logs are disposed of.	Logging	[46]
Evidence		
All possible evidence sources are identified.	Evidence	[54]
Evidence contains all necessary information to be classified as complete and valid.	Evidence	[33], [54]
All evidence data are secured.	Evidence, Encryption	[56], [57]
Evidence storage is reliable.	Evidence	[55]

a) Access Control: Access control ensures that data cannot be changed or read by unauthorized entities. This enhances their confidentiality and integrity, which are essential for software security [64].

b) Authentication: Authentication is a process of dedication and confirmation of the true identity of an external entity [42]. CI systems work with sensitive data, and their functionalities are safety-critical; therefore, access to them should be limited to trusted people and components.

c) Authorization: Authorization is deciding if a concrete authenticated entity is allowed to execute or make a specific action. Users and devices should have various roles based on their permission and have minimal possible rights [65], [39].

d) Data Protection: Data are often the main target of cyberattacks. They have to be protected from theft, unauthorized changes, or corruption [37].

e) Logging: Logging is the fundamental method to control and collect information about the program's behavior. It can help us detect an ongoing attack, gather evidence, improve the development process, etc. Therefore the whole mechanism is quite complex and should not be underestimated [47], [66].

f) Anomaly Detection: Anomaly detection means searching for deviations from the expected behavior of the system. All found anomalies should be inspected because they could signify an incident or danger. Correct detection can be tricky; possible false positives and negatives can be disastrous [48].

g) Input and Output (I/O): I/O vulnerabilities are not only frequent targets of attacks but also weak points vulnerable to inadequate usage by ordinary users. Operating a system without proper I/O handling is like leaving a house with doors and windows open [49].

h) Phenomenon Evaluation: This category covers a thin layer between anomaly detection and reaction to the incident. The reaction should not be reckless and hasty and must be evaluated adequately to the severity of the detected anomaly.

i) Network: With availability improvement and new technologies it is not imaginable to operate a CI system without any network connection anymore [67]. That brings up a variety of potential problems and vulnerabilities [68].

j) Stopping from Propagating: There can be many weak points across the software, and attackers may want to gain control of the entire system from them. Therefore we have to stop the spreading of any incident so it will not be possible for a localised fault to endanger distant parts of the system.

k) Safety Ensuring: Safety is an essential property of CI systems. As it strongly depends on the CI domain, we provide only an abstract solution to its ensuring within the system. This tag also indicates guidelines in other categories that may affect system safety.

l) Backups: System recovery depends directly on backups. Without proper backup, we may not be able to repair the system, and data will be lost. We must plan backups while designing the system because it would be too late to save data when an incident is detected. We also have to protect backups so they will not be damaged together with the system during the incident [69].

m) Evidence: Evidence is a cornerstone of forensic readiness. Identifying and collecting pieces of evidence should be taken into account already during the design of the system, not during the incident [70]. Evidence does not come by itself; we must be prepared to collect it and maximize its quality to facilitate the investigation [54], i.e. to ensure its integrity, prevent leaks, and protect contained sensitive data.

n) Third-Party Components: Third-party components often do not have as strict quality requirements as CI systems. Due to this, they may have safety issues that can propagate to our system. Therefore third-party components should be observed and not trusted by default [71].

o) Encryption: Encryption helps us preserve the integrity and confidentiality of the data within the system [72]. Similar to Network, this tag is mainly intended to identify encryption-related guidelines. Choice of concrete encryption techniques is out of the scope of architectural considerations.

p) Self-Adaptivness: CI systems may be too complex to be managed entirely by humans. Self-adaptive software monitors itself and its environment and reacts appropriately to detected changes. Therefore, such a system will be easier to maintain, and its responses to incidents will be faster [73].

VI. Conclusion

In this paper, we have presented a vision of an integrated checklist guiding the design of critical software systems, and presented first version of such a checklist. To this end, we did research to collect relevant standards and sources, all covering the scope only partially, and proposed a set of guidelines in the form of a checklist to enhance its straightforward usability during the software design. Guidelines were classified and sorted out to meet our defined checklist scope. Additionally, supplementary material is available at [7], containing the full guidelines classification data, detailed guidelines descriptions, and a demonstration of the checklist in a real-life context. In the future, we would like to validate the checklist with industrial experts and refining it with further views and layers of detail.

References

[1] I. Meedeniya, A. Aleti, and B. Buhnova, "Redundancy allocation in automotive systems using multi-objective optimisation," in *Symposium of Avionics/Automotive Systems Engineering (SAASE'09), San Diego, CA*, 2009.

[2] S. Chren, B. Rossi, B. Bühnova, and T. Pitner, "Reliability data for smart grids: Where the real data can be found," in *2018 smart city symposium prague (scsp)*. IEEE, 2018, pp. 1–6.

[3] J. Rodríguez, A. Galán, A. Alvarez, R. Díaz, and C. Consortium, *D2.1, CIPSEC System Design WP 2, Development of the CIPSEC security framework for Critical Infrastructure environments CIPSEC Enhancing Critical Infrastructure Protection with innovative SECurity framework*, Jan 2017.

[4] E. The European Union Agency for Cybersecurity, "Eu cybersecurity certification framework," Dec 2020. [Online]. Available: https://www.enisa.europa.eu/topics/standards/certification

[5] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice, 3rd edition*. Addison-Wesley Professional, 2013.

[6] E. Fernandez-Buglioni, *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons, 2013.

[7] A. Bierska, B. Buhnova, and H. Bangui, "Supplementary material for the integrated checklist," https://drive.google.com/drive/folders/1DNjQdBmVTR7Z_JYZYpQS_QaohdFboIkC?usp=sharing, 2022.

[8] K. Lukitsch, M. Müller, and C. Stahlhut, "Criticality," in *Key Concepts for Critical Infrastructure Research*. Springer, 2018, pp. 11–20.

[9] "Federal ministry of the interior, national strategy for critical infrastructure protection, berlin, germany (www.bmi.bund.de, 2009."

[10] N. Medvidovic and R. N. Taylor, *Software architecture: foundations, theory, and practice*. John Wiley & Sons, 2010.

[11] G. Fairbanks, *Just enough software architecture: a risk-driven approach*. Marshall & Brainerd, 2010.

[12] M. Fowler, *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[13] I. Jacobson, "The immense power of simple check-lists for monitoring projects," https://www.ivarjacobson.com/publications/blog/power-checklists, 2020.

[14] Z. A. Baig, "Multi-agent systems for protecting critical infrastructures: A survey," *Journal of Network and Computer Applications*, vol. 35, no. 3, pp. 1151–1161, 2012.

[15] C. Consortium, "D1.3, report on taxonomy of the ci environments," Feb 2018. [Online]. Available: https://www.cipsec.eu/sites/default/files/cipsec/public/content-files/deliverables/D1.3%20Report%20on%20Taxonomy%20of%20the%20CI%20environments.pdf

[16] B. Buhnova, T. Kazickova, M. Ge, L. Walletzky, F. Caputo, and L. Carrubbo, "A cross-domain landscape of ict services in smart cities," in *Artificial Intelligence, Machine Learning, and Optimization Tools for Smart Cities*. Springer, 2022, pp. 63–95.

[17] B. Robert, R. De Calan, and L. Morabito, "Modelling interdependencies among critical infrastructures," *International Journal of Critical Infrastructures*, vol. 4, no. 4, pp. 392–408, 2008.

[18] P. Cichonski, T. Millar, T. Grance, and K. Scarfone, "Computer security incident handling guide : Recommendations of the national institute of standards and technology," *Computer Security Incident Handling Guide*, vol. 2, Aug 2012. doi: 10.6028/nist.sp.800-61r2. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf

[19] C. M. Machuca, S. Secci, P. Vizarreta, F. Kuipers, A. Gouglidis, D. Hutchison, S. Jouet, D. Pezaros, A. Elmokashfi, P. Heegaard *et al.*, "Technology-related disasters: A survey towards disaster-resilient software defined networks," in *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*. IEEE, 2016, pp. 35–42.

[20] S. S. Murtaza, W. Khreich, A. Hamou-Lhadj, and A. B. Bener, "Mining trends and patterns of software vulnerabilities," *Journal of Systems and Software*, vol. 117, pp. 218–228, 2016.

[21] F. Kadri, B. Birregah, and E. Châtelet, "The impact of natural disasters on critical infrastructures: A domino effect-based study," *Journal of Homeland Security and Emergency Management*, vol. 11, no. 2, pp. 217–241, 2014.

[22] G. Kirov, P. Zlateva, and D. Velev, "Software architecture for rapid development of hla-integrated simulations for critical infrastructure elements under natural disasters," *International Journal of Innovation, Management and Technology*, vol. 6, no. 4, p. 244, 2015.

[23] L. N. Alrawi and T. Pusatli, "Investigating end user errors in oil and gas critical control systems," in *Proceedings of the 2020 6th International Conference on Computer and Technology Applications*, 2020, pp. 41–45.

[24] T. Plėta, M. Tvaronavičienė, S. D. Casa, and K. Agafonov, "Cyber-attacks to critical energy infrastructure and management issues: Overview of selected cases," 2020.

[25] T. Limba, T. Plėta, K. Agafonov, and M. Damkus, "Cyber security management model for critical infrastructure," 2019.

[26] P. J. G. Seoane, "Use and limitations of checklists. other strategies for audits and inspections," *The Quality Assurance Journal: The Quality Assurance Journal for Pharmaceutical, Health and Environmental Professionals*, vol. 5, no. 3, pp. 133–136, 2001.

[27] M. Sibbald, A. B. de Bruin, and J. J. van Merrienboer, "Checklists improve experts' diagnostic decisions," *Medical education*, vol. 47, no. 3, pp. 301–308, 2013.

[28] E. Verdaasdonk, L. Stassen, P. P. Widhiasmara, and J. Dankelman, "Requirements for the design and implementation of checklists for surgical processes," *Surgical endoscopy*, vol. 23, no. 4, pp. 715–726, 2009.

[29] North American Electric Reliability Corporation - NERC, "Critical infrastructure protection standards," 2011.

[30] "IEEE standard for software safety plans," *IEEE Std 1228-1994*, pp. 1–24, 1993. doi: 10.1109/IEEESTD.1993.9097571

[31] D. G. Photovoltaics and E. Storage, "IEEE standard for interconnection and interoperability of distributed energy resources with associated electric power systems interfaces," *IEEE Std*, pp. 1547–2018, 2018.

[32] "IEEE standard for intelligent electronic devices cyber security capabilities," *IEEE Std 1686-2013 (Revision of IEEE Std 1686-2007)*, pp. 1–29, 2014. doi: 10.1109/IEEESTD.2014.6704702

[33] "IEEE standard cybersecurity requirements for substation automation, protection, and control systems," *IEEE Std C37.240-2014*, pp. 1–38, 2015. doi: 10.1109/IEEESTD.2015.7024885

[34] N. Aeronautics and S. Administration, "Nasa-std-8719.13 software safety standard," 2020.

[35] National Aeronautics and Space Administration, "Nasa-std-8739.8 software assurance and software safety standard," 2020.

[36] N. I. of Standards and Technology, "Nistir 7628 revision 1 – guidelines for smart grid cybersecurity," *The Smart Grid Interoperability Panel – Smart Grid Cybersecurity Committee*, 2014.

[37] O. Tayan, "Concepts and tools for protecting sensitive data in the it industry: a review of trends, challenges and mechanisms for data-protection," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 2, pp. 46–52, 2017.

[38] M. Papaioannou, M. Karageorgou, G. Mantas, V. Sucasas, I. Essop, J. Rodriguez, and D. Lymberopoulos, "A survey on security threats and countermeasures in internet of medical things (iomt)," *Transactions on Emerging Telecommunications Technologies*, p. e4049, 2020.

[39] B. W. Lampson, "Computer security in the real world," *Computer*, vol. 37, no. 6, pp. 37–46, 2004.

[40] E. B. Fernandez and J. Hawkins, "Determining role rights from use cases," in *Proceedings of the second ACM workshop on Role-based access control*, 1997, pp. 121–125.

[41] S. Mare, A. M. Markham, C. Cornelius, R. Peterson, and D. Kotz, "Zebra: Zero-effort bilateral recurring authentication," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 705–720.

[42] T. Nandy, M. Y. I. B. Idris, R. M. Noor, L. M. Kiah, L. S. Lun, N. B. A. Juma'at, I. Ahmedy, N. A. Ghani, and S. Bhattacharyya, "Review on security of internet of things authentication mechanism," *IEEE Access*, vol. 7, pp. 151 054–151 089, 2019.

[43] A. Naiakshina, A. Danilova, C. Tiefenau, M. Herzog, S. Dechand, and M. Smith, "Why do developers get password storage wrong? a qualitative usability study," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 311–328.

[44] M. Alsaleh, M. Mannan, and P. C. Van Oorschot, "Revisiting defenses against large-scale online password guessing attacks," *IEEE Transactions on dependable and secure computing*, vol. 9, no. 1, pp. 128–141, 2011.

[45] J. R. de Almeida, J. B. Camargo, B. A. Basseto, and S. M. Paz, "Best practices in code inspection for safety-critical software," *IEEE software*, vol. 20, no. 3, pp. 56–63, 2003.

[46] M. E. Whitman and H. J. Mattord, *Principles of incident response and disaster recovery*. Cengage Learning, 2021.

[47] A. Pecchia, M. Cinque, G. Carrozza, and D. Cotroneo, "Industry practices and event logging: Assessment of a critical software development process," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2. IEEE, 2015, pp. 169–178.

[48] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[49] J. K. Teto, R. Bearden, and D. C.-T. Lo, "The impact of defensive programming on i/o cybersecurity attacks," in *Proceedings of the SouthEast Conference*, 2017, pp. 102–111.

[50] M. Maass, A. Sales, B. Chung, and J. Sunshine, "A systematic analysis of the science of sandboxing," *PeerJ Computer Science*, vol. 2, p. e43, 2016.

[51] A. Tarinejad, H. Izadkhah, M. M. Ardakani, and K. Mirzaie, "Metrics for assessing reliability of self-healing software systems," *Computers & Electrical Engineering*, vol. 90, p. 106952, 2021.

[52] A. Mattavelli, "Software redundancy: what, where, how," Ph.D. dissertation, Università della Svizzera italiana, 2016.

[53] E. Nemeth, G. Snyder, S. Seebass, and T. Hein, *UNIX system administration handbook*. Pearson Education, 2000.

[54] R. Rowlingson *et al.*, "A ten step process for forensic readiness," *International Journal of Digital Evidence*, vol. 2, no. 3, pp. 1–28, 2004.

[55] L. Pasquale, D. Alrajeh, C. Peersman, T. Tun, B. Nuseibeh, and A. Rashid, "Towards forensic-ready software systems," in *2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)*. IEEE, 2018, pp. 9–12.

[56] J. McQuaid, "Forensic considerations for cloud data storage - forensic focus," 2021. [Online]. Available: https://www.forensicfocus.com/webinars/forensic-considerations-for-cloud-data-storage/

[57] A. Singh, R. A. Ikuesan, and H. Venter, "Secure storage model for digital forensic readiness," *IEEE Access*, vol. 10, pp. 19 469–19 480, 2022.

[58] M. Hollick and S. Katzenbeisser, "Resilient critical infrastructures," in *Information Technology for Peace and Security*. Springer, 2019, pp. 305–318.

[59] M.-D. McLaughlin and J. Gogan, "Challenges and best practices in information security management," *MIS Quarterly Executive*, vol. 17, no. 3, p. 12, 2018.

[60] F. C. Freiling and B. Schwittay, "A common process model for incident response and computer forensics," *IMF 2007: IT-Incident Management & IT-Forensics*, 2007.

[61] E. C. Thompson, *Cybersecurity incident response: How to contain, eradicate, and recover from incidents*. Apress, 2018.

[62] R. Fateman, "Software fault prevention by language choice: Why c is not my favorite language," in *Advances in Computers*. Elsevier, 2002, vol. 56, pp. 167–188.

[63] A. Valjarevic and H. S. Venter, "Harmonised digital forensic investigation process model," in *2012 Information Security for South Africa*. IEEE, 2012, pp. 1–10.

[64] Q. He and A. I. Antón, "Requirements-based access control analysis and policy specification (recaps)," *Information and Software Technology*, vol. 51, no. 6, pp. 993–1009, 2009.

[65] K. Walsh, "Authorization and trust in software systems," 2012.

[66] G. Rong, Q. Zhang, X. Liu, and S. Gu, "A systematic review of logging practice in software engineering," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2017, pp. 534–539.

[67] L. A. Maglaras, K.-H. Kim, H. Janicke, M. A. Ferrag, S. Rallis, P. Fragkou, A. Maglaras, and T. J. Cruz, "Cyber security of critical infrastructures," *Ict Express*, vol. 4, no. 1, pp. 42–45, 2018.

[68] G. Tzokatziou, L. Maglaras, and H. Janicke, "Insecure by design: Using human interface devices to exploit scada systems," in *3rd International Symposium for ICS & SCADA Cyber Security Research 2015 (ICS-CSR 2015) 3*, 2015, pp. 103–106.

[69] M. M. Howell, "Data backups and disaster recovery planning," 2003.

[70] L. Daubner, M. Macak, B. Buhnova, and T. Pitner, "Verification of forensic readiness in software development: A roadmap," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 1658–1661.

[71] D. E. Rico and M. Hann, "A combined dependability and security approach for third party software in space systems," *arXiv preprint arXiv:1608.06133*, 2016.

[72] J. Obert, P. Cordeiro, J. T. Johnson, G. Lum, T. Tansy, N. Pala, and R. Ih, "Recommendations for trust and encryption in der interoperability standards," Sandia National Lab.(SNL-NM), Albuquerque, NM (United States); Kitu Systems, Tech. Rep., 2019.

[73] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM transactions on autonomous and adaptive systems (TAAS)*, vol. 4, no. 2, pp. 1–42, 2009.