The Journal of Systems and Software 211 (2024) 111989

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Design for dependability — State of the art and trends ☆

Hezhen Liu [a], Chengqiang Huang [a,*], Ke Sun [a], Jiacheng Yin [a], Xiaoyu Wu [a], Jin Wang [a], Qunli Zhang [a], Yang Zheng [a], Vivek Nigam [b], Feng Liu [b], Joseph Sifakis [c]

[a] Huawei Technologies Co., Ltd., Bantian, Shenzhen, China
[b] Huawei Technologies Co., Ltd., Riesstr. 25, Munich, Germany
[c] Verimag, Université Grenoble Alpes, Grenoble, France

ARTICLE INFO

Keywords:
Design for dependability
Risk analysis
Risk mitigation
Risk assessment
Run-time assurance
Dependable AI systems

ABSTRACT

This paper presents an overview of design for dependability as a process involving three distinct but interrelated activities: risk analysis, risk mitigation, and risk assessment. Although these activities have been the subject of numerous works, few of them address the issue of their integration into rigorous design flows. Moreover, most existing results focus on dependability for small-size safety-critical systems with specific static architectures. They cannot be applied to large systems, such as autonomous systems with dynamic heterogeneous architectures and AI components. The overwhelming complexity and lack of interpretability of AI present challenges to model-based techniques and require empirical approaches. Furthermore, it is impossible to cope with all potential risks at design time; run-time assurance techniques are necessary to cost-effectively achieve the desired degree of dependability. The paper synthesizes the state of the art showing particularly the impact of new trends stemming from the integration of AI components in design flows. It argues that these trends will have a profound impact on design methods and the level of dependability. It advocates the need for a new theoretical basis for dependability engineering that allows the integration of traditional model-based approaches and data-driven techniques in the search for trade-offs between efficiency and dependability.

1. Introduction

Systems engineering requires a systematic and rigorous set of processes applied iteratively for the design, operation and maintenance of systems throughout their life cycle (ISO/IEC/IEEE 15288:2015, 2015; SEBoK Editorial Board, 2022). System design aims at building cost-effective systems that meet given technical requirements. Regardless of the adopted methodology, systems engineers distinguish between system design aiming correctness under nominal conditions, and design for dependability aiming at making the system resilient to various kinds of hazards (Kececioglu, 1991; Stapelberg, 2009; Patrick O'Connor, 2012; Day et al., 2012). This distinction reflects a principle of separation of concerns in systems engineering that is important for coping with the overall system design complexity. For instance, for a flight controller, nominal conditions may assume that all the electromechanical devices onboard perform as expected. After the design of the basic functions of the controller under nominal conditions, dependability analysis will focus on identifying the possible violations of nominal conditions and understanding their effects to facilitate the enhancement of the original design.

Note that design for nominal correctness is top-down and aims to build a system that meets given requirements. A typical example of such a design flow is the V-model (Kevin Forsberg, 2005; ISO 26262-1:2018, 2018) that prescribes steps leading from requirements to an architecture, the detailed design of components and their progressive integration accompanied by an associated validation process.

Design for dependability is bottom-up and focuses on making the designed system resilient to hazards arising from different types of risk factors including failures, design flaws, disturbances from the physical environment and attacks. It is probably the hardest kind of design as it requires a deep and global knowledge of the system's functional behavior, its implementation and above all its interaction with the external environment. It also requires good common-sense engineering skills to estimate what can go wrong and how frequently it can happen using all available evidence, primarily past experience and expert judgment. In contrast to the concept of nominal correctness, dependability is not a "black or white" concept. It expresses assurance that the designed system can be trusted that it would perform as expected despite any kind of hazards resulting from violations of the

☆ Editor: Hongyu Zhang.

* Corresponding author.
E-mail addresses: liuhezhen2@huawei.com (H. Liu), huangchengqiang@huawei.com (C. Huang).

https://doi.org/10.1016/j.jss.2024.111989
Received 14 September 2023; Received in revised form 25 December 2023; Accepted 2 February 2024
Available online 5 February 2024

nominal conditions. It is characterized by a set of attributes such as reliability, availability, maintainability, etc. which are by their nature probabilistic/non-deterministic, reflecting the lack of predictability of the various risk causes. All these attributes characterize the ability of a system to provide the expected service without distinguishing which types of properties are affected by the hazards. Design for dependability involves three distinct but interrelated activities:

1. *risk analysis* that aims at linking risk causes to their effects on system behavior;
2. *risk mitigation*, which, based on the results of risk analysis, develops risk detection, isolation and recovery mechanisms integrated into fault-tolerant architectures;
3. *risk assessment* to estimate dependability attributes, e.g., reliability, availability, and maintainability for the designed system.

Each one of these activities has been the object of a great number of works. Nonetheless, only few address the issue of their integration into rigorous design flows. Existing results focus on dependability for small-size safety-critical systems with specific static architectures. They are not applicable to large critical systems such as autonomous systems with dynamic heterogeneous architectures and integrating AI-enabled components. For such systems, the number of risk causes is overwhelming. Connecting them to their effects requires more powerful analysis techniques. Furthermore, it is not possible to cope with all potential risks at design time; run-time assurance techniques are necessary for achieving cost-effectively a desired degree of dependability. Finally, an additional obstacle to the unification among the various approaches is the lack of agreement on a common terminology identifying a minimal set of key concepts and their relations. Significant differences between terminologies, adopted by standard committees and research communities, result in a conflation of terms and confusion, e.g., the work of Avizienis et al. (2004).

In this paper, we present a critical assessment of the state of the art on design for dependability and propose avenues toward a unified design flow. We advocate integration of existing results in such a flow and identify gaps in the current state of the art that should be filled to meet current systems engineering needs.

Our analysis takes into account the following trends in systems engineering:

We need better integration of design for dependability in the system development lifecycle, strongly supported by tool automation. Dependability Engineering has historically emerged as a sub-discipline of Physical Systems Engineering and is often presented as something apart from the other system design activities. The rich existing literature rarely addresses the issue of their integration into rigorous design flows. Most work focuses on building small centralized safety-critical systems with static architectures of guaranteed dependability, characterized by a set of quantitative attributes such as reliability, availability, maintainability, serviceability, manageability, etc. Works on security-critical systems, such as smart cards, SCADA systems, and networks, developed much later. They adopted different approaches that are less statistical in nature given that human action (non-statistically predictable) is the risk cause. However, with the advent of industrial IoT and autonomous cars, both safety and security should be considered together as they can impact each other. For instance, a security vulnerability in a connected car could be used to disable braking while driving, resulting in potential loss of control and a crash.

We need hybrid design techniques that allow the integration of the traditional model-based approaches and data-based ones using AI-enabled components. We know that neural networks cannot be trusted as their behavior is not explainable. They are however essential for efficiently solving computationally hard problems including especially perception and adaptation for autonomous systems. The challenge is to build reliable systems from unreliable components, an old problem successfully solved for fault-tolerant hardware systems using redundancy. To solve this problem, we advocate the deployment of monitoring mechanisms that rely on knowledge of desired system properties and reconfiguration techniques to adapt to hazardous situations.

Systems engineering is currently at a turning point. We have successfully applied design for reliability to simple, centralized, and automated systems with predictable environments and explainable specifications such as avionic systems, nuclear plants and production systems. New techniques should be applicable to complex, decentralized, autonomous systems with unpredictable environments and non-elicitable specifications. Such a change may render obsolete rigorous techniques relying on mathematical models and mark a shift toward more empirical techniques based on simulation and testing.

The paper is structured as follows. Section 2 discusses how design for dependability can be integrated into a general system design flow. Section 3 provides an overview of the three main activities of design for dependability, and discusses how current trends may affect the underlying methods and practices. Section 4 concludes by highlighting the need for new theoretical foundations for dependable intelligent systems, bridging the gap between traditional model-based and data-driven AI techniques.

2. Design for dependability in a general system design flow

System design is the process leading from given technical requirements to a validated system meeting pre-established acceptance criteria (ISO/IEC/IEEE 15288:2015, 2015; SEBoK Editorial Board, 2022). The technical requirements are the result of the analysis of stakeholders' expectations. They include functional requirements specifying the provided functionality usually provided by software components and extra-functional requirements including efficiency and dependability criteria. Furthermore, technical requirements clearly define *nominal conditions* of system operation including assumptions about the execution environment as well as interactions with potential users and managers.

System correctness is generally expressed as a set of properties resulting from the analysis of nominal requirements. A system is judged correct or not according to the satisfaction of these properties as nominal requirements often ignore imponderable factors. Correctness deals with three types of properties:

1. *safety properties* that express that nothing bad happens, that is the state of the designed system will never satisfy unsafe conditions determined from analysis of the requirements;
2. *security properties* that express that the designed system is resilient to attacks and malevolent actions under nominal conditions;
3. *efficiency properties* that characterize the useful work provided by the system depending on resource availability (memory, energy, time, money) subject to two often conflicting criteria: *performance* and *cost*. Performance describes how well the system does with respect to user demands, e.g., throughput, jitter, latency, and quality. Cost measures how well resources are used with respect to economic criteria, e.g., storage efficiency, energy efficiency, processor utilizability.

From a methodological point of view, we consider that system design consists of two main tasks. The first task leads to an implementation that is correct under nominal conditions. It starts with the development of an architecture reflecting a logical decomposition of the functional requirements into corresponding subsystems and components with associated coordination mechanisms. This is shown in the design for functionality section in Fig. 1 (box I). Then after verification and validation of the function, the obtained functional architecture is deployed on execution platforms. This process is shown in the design for performance section in Fig. 1 (box II). The corresponding implementations are evaluated, e.g., using design space exploration techniques, to select the most adequate design with respect to system efficiency requirements.

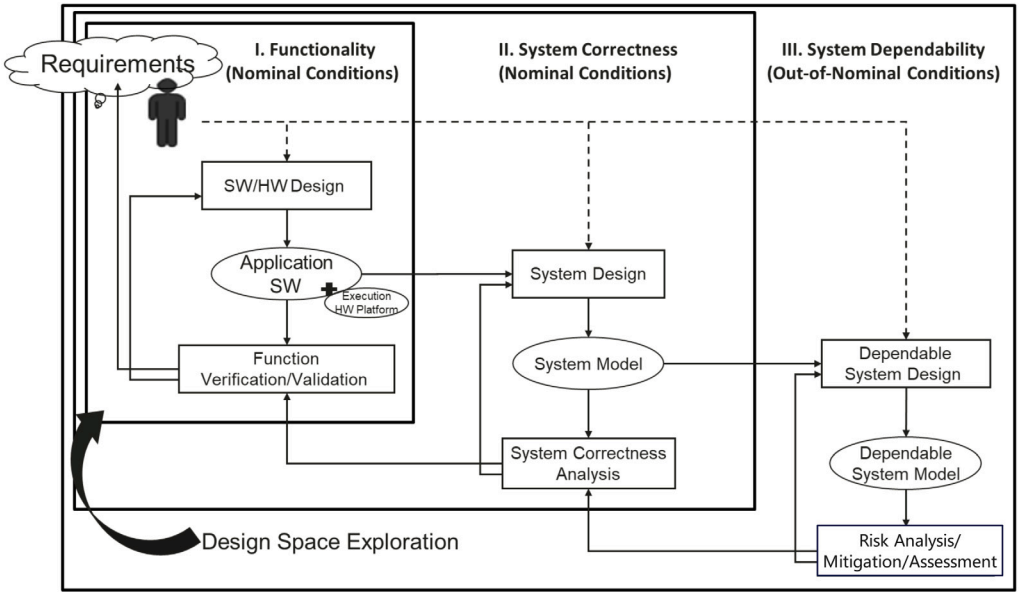

Fig. 1. The separation of concerns in system design flow.

The second task of system design deals with meeting dependability requirements (box III in Fig. 1). As explained in the Introduction section, it combines risk analysis, design of risk mitigation mechanisms, and assessment of dependability attributes. We will detail these in the next section.

It is worth noting again that the key difference between the two design tasks is that their standpoints are radically different. The task for system functionality and performance considers the correctness of the system under nominal conditions. The nominal conditions could be extreme for some systems. For example, the nominal working temperature for a spaceship could be ranging from −150 degree centigrade to 150 degree centigrade. For the spaceship to operate reliably, a thermal protection coating and the underlying mechanisms are part of the design for functionality, as it is a mandatory element for spacecraft operation.

On the contrary, the task for system dependability is to look for possible violations of nominal conditions due to various risk factors. "What if" questions are often raised. For example, what happens if one of the engines in a civilian aircraft fails? Such situations require a specific analysis taking into account all the safety characteristics of the system and determining the adequate mechanisms to ensure its operational dependability. In the remainder of the paper, we provide an overview of risk analysis, mitigation, and assessment methods and review key trends in the state of the art.

3. Design for dependability methodology

Historically design for dependability has followed two different avenues depending on the nature of risk causes. One seeks safety that is resilience against risk not caused by intentional human action, such as component failures, programming errors, and physical environment disturbances; the other seeks security that is resilience to malevolent actions including attacks of any kind.

The two avenues have followed quite separate approaches with different contexts, constraints, and timelines. Safety-critical systems are often reactive and subject to strong real-time constraints, e.g., avionic, nuclear, and railroads, while security-critical systems are interactive, e.g., smart cards, core banking systems. Safety standards propose well-defined safety assurance processes and have been defined for the different industrial sectors, while the landscape of security standards is very large and less structured. Safety requirements analysis and elicitation are much easier than security requirements. Formalizing global security properties is practically impossible because adversity comes from the human factor whose malicious behaviors cannot be predicted by modeling, for instance, in meltdown attacks.

Safety and security must be considered together when designing a modern system. Modern vehicles have multiple potential vulnerabilities, known as attack surfaces: integrated Wi-Fi, cellular, Bluetooth, onboard diagnostics, USB, and other connection points all provide potential routes into the vehicle communication systems. Each of these features should be made resilient to attacks by design. Both dependability for safety and dependability for security share similar flows involving (1) risk analysis; (2) risk mitigation; and (3) risk assessment. While the root causes of the problems vary according to the type of property, there are important commonalities in the treatment of the effects, i.e., the hazards from which the system must be protected regardless of their origin.

As we have explained, dependability is generally characterized by the quantitative attributes represented by probabilities. However, probabilistic modeling cannot be applied to attacks and their effects, as they are not statistically predictable. It does not make sense to consider that reliability accounts for security aspects. Some security vulnerabilities are present at design time and others emerge over a lifetime. The exploitation of vulnerabilities depends on human creativity and the evolution of technology. Consequently, security assessment is by nature more qualitative, and threat mitigation should be a continuous concern.

In this section, we present dependability techniques that focus on the analysis and mitigation of hazards regardless of their cause. However, most risk quantification and analysis methods are applicable to safety-related hazards, which also form the basis of the traditional design for dependability methodologies.

3.1. Risk analysis

3.1.1. Basic terminology

Risk analysis starts with the identification of potential risk causes that are plausible violations of nominal conditions. Risk causes can be artificial or natural. *Artificial causes* may be intentional, e.g., malevolent actions, attacks, or non-intentional, e.g., programming or design defects, misuses, etc. *Natural causes* include failures of peripheral devices and disturbances from the physical environment. This distinction is important because natural causes may be characterized by random

variables, e.g., following a distribution determined by aging or fatigue laws of materials. On the contrary, causes of human origin are much more difficult to analyze. Another important distinction for risk analysis is between internal and external causes. *Internal causes* occur in the designed system just because it fails to satisfy by design some technical requirement while external causes occur in the system environment.

As noted above, the terminology used to distinguish between different risk factors has been the subject of specific work and analysis (Avizienis et al., 2004). For the purposes of this paper, we will use minimal terminology assuming that the precise meaning of a term is clear from the context of use.

We consider a "fault" to be the most basic non-deliberate cause of risk, such as a defect, imperfection or flaw that can cause problems. Faults can have a variety of sources, including the system environment, the software or hardware of an actual system implementation, or its design. Faults can be discovered by independent analysis to determine if they are latent or active. We use the term "error" to denote a fault that may violate an essential system property or specification, such as programming or design errors. On the contrary, we use the term "failure" to denote a fault that may imply the inability to provide the expected result at an interface. A failure is therefore closely related to a component of a system. For example, a crack in the pistons of an engine is a fault that can cause an engine failure. Finally, we call hazard any violation of nominal requirements regardless of its origin deliberate or not. So a hazard can be a manifestation of faults or attacks exploiting system vulnerabilities. Taking autonomous driving as an example, the purpose of risk analysis is to determine the risk causes and their possible impacts on the overall system behavior (Apostolakis, 2004). We are faced with failures, such as a flat tire or engine failure, which can lead to various safety hazards violating nominal system properties. Design for dependability must identify the various causes of risk and implement risk mitigation mechanisms to ensure safety with a very high probability.

3.1.2. Risk analysis techniques

Risk analysis techniques provide methodologies for generating scenarios that relate risk causes to hazards. Note that the generation process cannot be fully automated because it implies the elicitation of relations between the system and its environment that are hardly amenable to formalization. For instance, finding the impact of the failure of a peripheral requires a deep knowledge of the system's functional behavior and implementation. The probabilities of these scenarios are estimated using all available evidence, primarily past experience and expert judgment. Finally, the hazardous scenarios are ranked according to their expected frequency of occurrence. This section presents several analysis methods that are well developed and widely used to design dependable systems.

3.1.2.1. Hazard Identification (HAZID). HAZID is a commonly used technique based on structural risk analysis to systematically identify hazards (Stapelberg, 2009). Applied in the initial phase of product development, it provides a worksheet of the set of causes, effects, and severities along with the identified hazards. Some of the HAZID norms are described in the military standard MIL-STD-882 and its descendants, which were developed by the U.S. Department of Defense to ensure system safety. The standard specifies the following contents in the HAZID table: (a) hazards and associated risks; (b) hazard-related functions, items, and materials; (c) operation, maintenance, sustainment, and disposal requirements; (d) preventive or corrective actions. While the integrity of the analysis depends on the expertise of the specialists, HAZID provides meaningful results from limited specification information. Numerous works have also sought to improve HAZID. For instance, Paltrinieri et al. (2013) present a novel systemic risk analysis technique that improves risk identification abilities in atypical scenarios for HAZID. Park (2017) proposes a method that extends conventional HAZID based on semi-automatic data processing for safety design.

3.1.2.2. Failure Mode and Effects Analysis (FMEA). FMEA is a risk analysis method that aims to list all failure modes that can cause hazards. It is typically subdivided to design FMEA (DFMEA) and process FMEA (PFMEA) (Sharma and Srivastava, 2018). DFMEA focuses on reliability analysis over the design phase of the system, while PFMEA is concerned with the reliability analysis of the system during production and operation. If quantitative criticality factors are considered for failure mode prioritization, a semi-quantitative FMEA, known as Failure Mode, Effects & Critical Analysis (FMECA), is performed. Since late 1940s, FMECA has been widely implemented for risk analysis in industries such as energy and manufacturing (Segismundo and Augusto Cauchick Miguel, 2008; Tazi et al., 2017).

However, FMEA has limitations in its applicability, cause and effect presentation, analysis exactness, and problem-solving breadth (Spreafico et al., 2017). Over the years, researches have been done to overcome the limitations. For example, Price and Taylor (2002) introduce approximate failure rates for components during FMEA. This allows automatic generation of all possible failures with quantified occurrence rates, and most possible failure modes of the system can be identified. Traditionally, FMEA focuses on enumerating failure modes from hazards, but neglects the logic of propagation among them. Lee (2001) suggests a hybrid Bayesian Belief Network FMEA approach, whereby spreadsheets of failure modes are transformed into model components. The causal relationships in between are established using a syntax based on conditional probabilities. The resulting Bayesian network-like diagram greatly enhances the capacity and applicability of FMEA.

3.1.2.3. Fault Tree Analysis (FTA) and Event Tree Analysis (ETA). FTA and ETA are both techniques used in risk analysis, but they have different focuses. FTA is a deductive method that starts with a hazardous event and works backwards to identify the root causes or contributing factors that led to the event. On the other hand, ETA is an inductive method that starts with a hazard and works forward to identify the potential consequences or outcomes of the hazard. Both techniques are useful in identifying and understanding how risks propagate in complex systems. Over the years, researchers had also managed to use FTA and ETA together to provide a more comprehensive analysis of safety and risk. As shown in Fig. 2, the Bow-tie model links the top event of FTA with the initial event of ETA. It provides a full picture for practitioners to investigate both the root causes and the consequences of hazard events (Čepin, 2011; Ju, 2016; Markowski and Kotynia, 2011).

3.1.2.4. System Theoretic Process Analysis (STPA). The increasing complexity of software-intensive systems has made traditional bow-tie safety analysis insufficient to identify hazardous situations due to emergent properties resulting from component composition (Leveson, 2019). To address this limitation, Leveson and Thomas (2018) have proposed STPA which is a risk analysis methodology focusing on unsafe interaction between components. The methodology involves five well-defined steps including: (1) the identification of high-level hazards that the analysis aims to prevent; (2) building the system's hierarchically structured model, specifying how the different functions interact with each other; (3) analysis of the impact of unsafe control actions of each component; (4) identifying where unsafe control actions can occur in the system by creating explanatory scenarios; and (5) proposing safety requirements to address such causes. A key feature of STPA is its emphasis on defining, using contextual variables, the situations in which systems operate (Thomas, 2013). A situation is a particular instance of the context variables. As emergent properties often depend on the situation in which the system is, STPA can identify scenarios, i.e., particular situations in which these emergent properties are not safe, even if no function fails. A problem, however, is that the number of situations may explode (exponentially) as one increases the types of contexts. Some approaches using Formal Methods have been proposed to address this problem (Thomas, 2013).

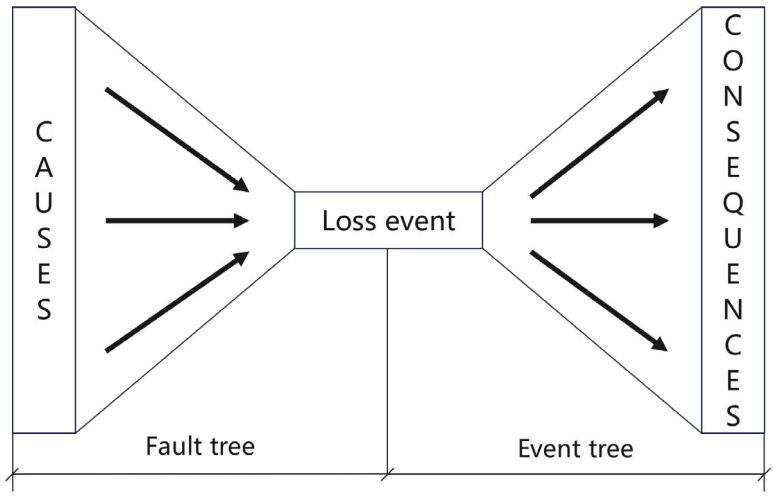

Fig. 2. An illustration of a bow-tie model (Markowski and Kotynia, 2011).

STPA has been applied to the risk analysis of safety-critical systems in the aeronautics industry. In recent years, with the increasing move towards software-defined vehicles, STPA finds application in the automotive industry to address the risk analysis of automated driving functions (ISO 21448:2022, 2022). Working groups, such as the SAE J3187 (SAE, 2022), are considering the adoption of STPA for functional safety (ISO 26262-1:2018, 2018).

3.1.2.5. Fault Propagation and Transformation Calculus (FPTC). FPTC and its variants (Wallace, 2005; Fenelon and McDermid, 1993; Delange et al., 2014) go one step further than STPA, in an attempt to analyze the causality relation between types of faults in hierarchical data-flow networks. In FPTC, components and their connectors are specified by an I/O relation, marked as "→", between two main types of values: (1) "$*$" denotes the absence of fault; (2) "f" is a variable denoting any type of fault that can be specialized by pre-defined sub-types. Hence, it is possible to write rules expressing for a component the fact that:

- $* \rightarrow f$, the component can produce faults when it receives correct inputs;
- $f \rightarrow *$, the component tolerates faults;
- $f \rightarrow f$, the component can propagate faults.

Works on FPTC propose fault ontologies defining possible subtypes of faults such as,

- *Value faults* (*Detectably wrong, Undetectably wrong*) when a component responds within the correct time interval but with wrong values;
- *Timing faults* (*Early, Late*) when a component responds with the correct value but outside the normal time interval;
- *Service provision faults* when a component fails to produce an appropriate output (*Omission*) or produces an inappropriate output (*Commission*).

FPTC provides an elegant framework for fault propagation. The faulty behavior is for each component a set of rules of the form $f_1 \rightarrow f_2$ where f_1 and f_2 are subtypes of faults. This gives a lot of modeling freedom. For acyclic data-flow architectures, the rules describing individual component behavior can be composed to characterize the system I/O faulty behavior. Rule composition may result in some combinatorial explosion due to the non-deterministic behavior of components. Nonetheless this can be easily automated. FPTC can be in principle extended to non-acyclic architectures by application of general data-flow analysis techniques. Note that FPTC is not applicable to static architectures that are not data-flow. Furthermore, the specification of the I/O faulty behavior of each component devolves to the human modeler.

Closing the overview of existing risk analysis techniques, we point out that they are applied to systems with static architectures and transformational components. Their application to complex systems such as autonomous systems is problematic because of their dynamism and the complexity of their environment. We advocate for these systems a more global approach that studies how faults and hazards affect the nominal system behavior as presented in Section 4.

3.2. Risk mitigation

3.2.1. Fault detection, isolation, and recovery techniques

Risk mitigation is traditionally achieved through the systematic combination of Fault Detection, Isolation, Recovery techniques (FDIR) (Hwang et al., 2010; Bittner et al., 2014; Zolghadri, 2017). These are techniques intended to mitigate risks by timely detecting their causes and isolating faults and canceling their effects. They rely on results of risk analysis that provides for each fault, associated sequences of faulty states causing the corresponding violation of system nominal correctness. They use mechanisms that contain the faulty behavior and possibly allow bringing back the system to some trustworthy state (Recovery). We provide below a description of the three types of FDIR techniques implemented using a variety of mechanisms depending on the type of the system and the sought degree of dependability.

3.2.1.1. Fault detection and prediction. Fault detection and prediction mechanisms are crucial for identifying potential issues in a system and triggering corrective actions. These mechanisms utilize various system data, including time-series and performance indicators, such as CPU metrics, logs and alarms.

One of the most common approaches to detect anomalies involves statistical methods. These are based on the principle that normal data generally occur in high-probability regions of a stochastic model, while anomalies occur in low-probability regions. These statistical-based methods comprise parametric methods that leverage univariate and multivariate outlier detection techniques based on the normal distribution (Rippel et al., 2021), as well as non-parametric methods that rely on other approaches, such as kernel density estimation (Hu et al., 2018). Furthermore, classification methods are widely used for anomaly detection by application of classification algorithms (Tian et al., 2014; Muniyandi et al., 2012) to train models capable of detecting outliers in unknown data. Other methods are based on clustering focusing on the relationship between objects and clusters to identify potential outliers (Chawla and Gionis, 2013; Cheng et al., 2019). Outliers represent objects that do not fit within any cluster.

Fault prediction, on the other hand, aims at forecasting equipment performance degradation and remaining life. Related methods can be divided into physical model-based methods and data-driven methods. Methods based on physical models generally require a mathematical model of the target system. Typical examples are like battery capacity prediction (Ashwin et al., 2018) and lifetime prediction of aircraft gas turbine engine compressor disk (Shlyannikov et al., 2020). Data-driven methods can analyze the system data without an accurate mathematical model. They include machine learning methods and statistical analysis methods. Machine learning methods include time series forecasting (Nguyen et al., 2021) and log series forecasting (Le and Zhang, 2022). Statistical analysis methods require a probability density function which can be obtained by analyzing the data history (Ma et al., 2019).

With the proliferation of machine learning and deep learning methods, fault detection and prediction have been made relatively easy under the situation that the quality and quantity of data are satisfied. However, when data is not suitable, data-driven methods may face tremendous difficulties. The development of hybrid solutions combining machine learning and model-based techniques, which require less modeling efforts, is an interesting avenue.

3.2.1.2. Fault isolation. Fault isolation involves organizing workloads to operate in separate fault domains (i.e., groups of components that share a single point of failure Microsoft contributors, 2023) to ensure that a malfunction in one component does not impact the entire system. Once a failure does occur, it remains contained within a specific domain, prevented from spreading to other sections of the system.

Fault isolation is implemented by allocating components to specific fault domains during the design, deployment, and operational stages of the system's lifecycle, which sometimes need the supports of fault detection and system monitoring techniques. Fault isolation methods can be classified into static and dynamic methods:

- Static Fault Isolation: this method involves assigning components to predetermined fault domains during the design and deployment phases. Typically for scenarios with static systems, achieving static fault isolation requires system designers to meticulously analyze system requirements, identify potential failure modes, and assess associated risks. Components are subsequently assigned to fault domains based on their dependencies, failure modes, and possible system impacts.
- Dynamic Fault Isolation: this method adjusts the fault domains according to the system's workload and its fluctuations. More challenging to implement compared to static fault isolation, it requires system designers to utilize real-time monitoring and fault detection techniques to discern changes in system conditions and workload distribution. They must also develop algorithms and policies for dynamically modifying fault domains based on these alterations.

There are a variety of fault isolation techniques applied to chips (Nagalingam et al., 2020), hardware (Liu et al., 2020), operating systems (Herder et al., 2009), and software (Peach et al., 2020). For example, in cloud service scenarios, failure domains may encompass larger areas such as data centers, cities or regions. Below, we present the main features of three classic fault isolation techniques:

- Linux namespace and cgroup techniques allow achieving operating system-level isolation (Joy, 2015). Namespaces are used to encapsulate global system resources within an abstraction layer, enabling processes within a namespace to access global resource instances (Proskurin et al., 2020). On the other hand, cgroups are used to restrict and manage system resources, preventing faults from propagating, such as out-of-memory (OOM) errors.
- A hardware-level fault isolation technique uses software to monitor and predict hardware malfunctions, proactively isolating potential addresses or fault blocks. This approach considerably reduces the probability of memory access errors (Costa et al., 2014).
- Incorporating microservices and distributed architecture during the design stage enables software-level fault isolation (Meynen et al., 2020). The system is divided into multiple autonomous service units that are coordinated and managed in a distributed manner, ensuring high availability and fault tolerance.

3.2.1.3. Fault recovery. Fault recovery generally relies on fault-tolerance mechanisms. We distinguish between static and dynamic fault tolerance schemes. Static fault tolerance schemes create robust and trusted composite operations at the design stage, such as high-reliability component service selection, component services ranking, exception handling, and transaction composition (Zhang et al., 2018). In contrast, a dynamic fault-tolerant scheme aims to try, repair and restart the actions of composite services at run-time, when components enter an error state. Typical examples of composite service actions in dynamic fault tolerance schemes include retry, task resubmission, and software rejuvenation. Techniques used in dynamic fault tolerance schemes can be divided into two broad categories: reactive and proactive techniques.

Reactive techniques are used to reduce the impact of failures on a system when the failures have actually occurred (Amin et al., 2015). They include checkpointing/restarting, replication, and retry.

- Checkpointing is the procedure of keeping the application status after any effective completion or storing the preliminary failure-free state. This method is implemented at various levels, including device level, application level, and library level. Mohammed et al. (2016) propose an effective fault tolerance strategy for long-term tasks and tools in cloud systems. In the event of a system failure, the system is configured to restart from the last safe and active checkpoint.
- Replication methods are generally used to improve system availability by using redundant resources. For example, to improve the availability and persistence of the data, the traditional design of the Hadoop Distributed File System (HDFS) uses replication to protect data. Each data block is replicated with multiple copies to provide double fault tolerance through e.g., pipeline replication, parallel replication, reconfigured lazy and enhancement lazy replication. Kaseb et al. (2019) advocate replication to overcome the limitations of the Redundant Independent Files system. This improves the recoverability of lost blocks (availability) and the ability of the system to continue operating in the presence of a lost block (reliability) with less computational overhead.
- Retry can be used to recover an operation from failure. In a retry, a failed task is executed from the beginning either at the same resource or at some other resources. In case of byzantine faults, retry can be carried out at the same resource. However, the number of retry attempts at the same resource may be limited according to the situation. For example, after five unsuccessful retry attempts on the same resource, the task may be migrated to some other resources.

Proactive methods track the system continuously and perform fault analyses to minimize the risk of faults before they appear. Fault prediction algorithms are continuously performed to determine the components' states so that the suitable mitigation method can be used to avoid failures. The main strategies used for proactive fault tolerance include software rejuvenation, self-healing, and preemptive mitigation etc.

- Kumari and Kaur (2021) suggest software rejuvenation in the form of a proactive fault tolerance strategy based on a fuzzy logic framework. Software rejuvenation is the concept of periodic or early shutdown of running software to clean up its internal state

and reset data structures. It can be used to solve the problem of aging devices without ever solving the underlying hardware problem.
- Self-healing techniques are capable of recovering from certain types of failure without any human intervention. In Nazari Cheraghlou et al. (2019), it is stated that these are the most appropriate for the fault-tolerant clouds.
- Levy et al. (2020) propose a preventive and adaptive failure mitigation service, NARYA, integrated in Microsoft's Azure compute platform. Narya is designed to predict imminent Azure host failures based on multi-layer system signals and also to decide smart mitigation actions. It also adopts a new approach to online experimentation (i.e., A/B testing Gui et al., 2015 and the multi-armed bandit method) to measure the impact of mitigation measures and adapt appropriately.

3.2.2. Fault tolerance by redundancy

In the previous sections, we presented an overview of FDIR activities, each contributing to fault mitigation. This section presents approaches in which fault tolerance is achieved by redundancy, in particular by duplicating elements in dedicated structures. Unlike approaches where failure mitigation is achieved by combining specific mechanisms, redundancy provides global solutions based on the assumption that failures in duplicated components occur independently. In other words, the probability of two duplicated elements failing simultaneously is extremely low. The idea of building reliable systems from unreliable components using redundancy goes back to von Neumann (1956). It has become the most fundamental technique in the industry and is applied in three different approaches:

- The first approach is by massive component replication or modular redundancy that consists of integrating many replicas of a component in an architecture. It uses adequate mechanisms to detect faults as discrepancies in component behavior and ensure correction using voting mechanisms. Double redundancy allows the detection of a single fault, while triple redundancy allows the correction of a single fault using a voting system that applies the majority principle. In general, a duplication of $2n+1$ elements allows to correct faults of multiplicity n. Modular redundancy finds wide application in real-time safety-critical systems in avionics and space, nuclear power plants and railway systems. It can protect against hardware faults for which the assumption of independent occurrence is valid. For software systems, duplication only makes sense if different versions of the same software are duplicated (Avizienis, 1985), which leads to expensive solutions without any guarantee that the independence assumption is verified. For this reason, massive redundancy techniques are less successful for software systems.
- The second approach is temporal redundancy, which achieves fault tolerance through the repeated execution of an operation such as the sending of a message or by checkpointing to replay a transaction as discussed in the previous section. It applies to faults whose causes are intermittent and whose repetition over time can lead to their detection and, with a little luck, to their correction.
- The third approach is information redundancy, which is achieved by replicating data or adding additional information to the minimally necessary information, so that a given relationship is maintained between the two, allowing errors to be detected once the relationship is violated. This principle is applied to error-correcting codes such as Hamming codes (Hamming, 1950), which are often used to cope with memory or transmission protocol failures. More rarely, arithmetic codes are used to cope with failures of arithmetic units (Massey and García, 1992) where the operands are expanded with their value modulo a prime integer.

These approaches find application in fault-tolerant architectures (Sorin, 2009) where a combination of them can be applied, e.g. modular redundancy and error correction codes. It is clear that simple resource redundancy is not enough to ensure fault tolerance. Care must be taken to ensure that fault management advantageously combines the integrated fault tolerant features. In addition, fault tolerance may be accompanied by a degradation in system performance or functionality. This requires solutions that seek trade-offs between criteria such as reliability, energy efficiency and cost.

While the idea of redundancy is not new, the field has seen new applications in industry and academia. From a hardware/software perspective, heterogeneous redundancy has gained popularity. In addition to traditional mission-critical systems, redundancy has found application in autonomous vehicles, with multiple sensors to improve perception reliability. In some operating systems, researchers discuss the plausibility of heterogeneous execution environments (Coppens et al., 2018) to prevent problems caused by system vulnerabilities. From a systems perspective, redundancy has found new applications in large-scale systems, e.g., cloud computing systems (Cheraghlou et al., 2016; Zheng et al., 2010), in medium-scale systems, e.g., the container system (Zhou and Tamir, 2022), and also in small-scale systems, e.g., a chip (Kadri and Koudil, 2019). In all of these systems, new research is striving to provide better solutions using off-the-shelf redundant resources. How to prepare, schedule, and adapt tasks to redundant resources become crucial to reduce system costs, e.g., in terms of performance, and to achieve benefits, e.g., in terms of reliability. Essentially, fault tolerance is simply a byproduct of system design leading to an architecture that naturally makes it immune to certain failures while meeting essential functional requirements.

In addition to the aforementioned research areas on fault tolerance, the robustness of neural networks has received a lot of attention in the last decade, with redundancy playing a key role in achieving this property. Neural networks use residual layer connections, e.g., in residual neural networks, and redundant neurons, e.g., the dropout strategy, to achieve a robust analysis of the input. In addition, different fault-tolerant neural network architectures (Liu et al., 2019) have also been proposed to improve the reliability of a neural network. With the proliferation of artificial intelligence applications, this field is becoming increasingly important and has already seen successful research results. It should continue to flourish and bring long-awaited results for a better reliability of neural networks.

3.2.3. Run-time assurance techniques

FDIR techniques have been successfully applied to small, centralized critical systems (Hwang et al., 2010; Bittner et al., 2014; Zolghadri, 2017; Müller et al., 2020). They focus on ensuring dependability at design time. They rely on worst-case analysis of all the risky situations. For each situation, the resources needed for safe or secure operation are statically reserved. As a result, the amount of physical resources may be some orders of magnitude higher than necessary. This may incur high production costs as well as increased energy consumption. For example, response times of a critical real-time system must be guaranteed to be less than a given deadline. These are computed from safe approximations of the worst-case execution times of its tasks that may be many orders of magnitude larger than the real execution times. The resulting hardware platforms for such systems are often overprovisioned.

The use of massive redundancy techniques entails high development and operational costs. Current trends in systems engineering call into question the applicability of FDIR techniques for two main reasons:

1. One is the overwhelming complexity and uncertainty of system environments due to the openness needed for interaction and access to massive data. This is readily apparent in autonomous system designs limited by aleatory, systemic and computational complexity factors.

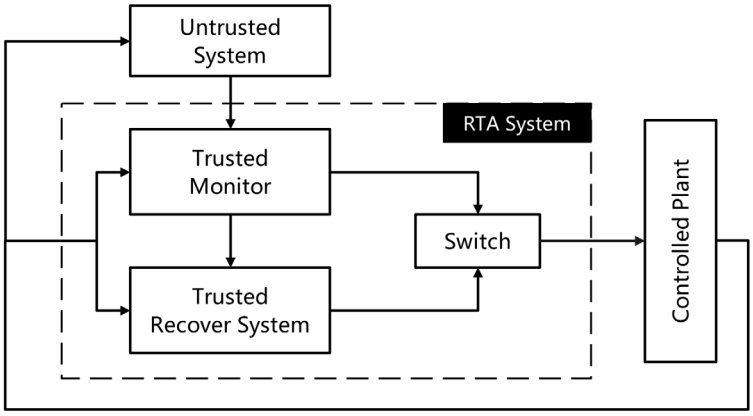

Fig. 3. The principle of Run-time Assurance.

2. The other reason is the explosion of risk causes where both system safety and security are at stake. Security risk analysis is usually much more involved than safety risk analysis. It is not just about the propagation of risk causes; it also requires figuring out possible attacks and anticipating intruder creativity.

We need a new approach that jointly addresses the various aspects of system dependability; that gives up the idea of exhaustive analysis at design time to determine hazards and develop specific mitigation mechanisms. Instead, the new approach should pursue the enforcement of essential global properties as much as possible independently of their potentially large number of causes. It should not rely on massive redundancy by duplication of components but instead, use monitors that at run-time detect deviations from nominal behavior and activate mitigation processes. Furthermore, it should extensively use the knowledge acquired at design time or at run-time to enhance predictability and anticipate high-risk situations and plan corresponding counter measures (Sifakis, 2018).

We present in the next paragraph the principle of a run-time assurance architecture that enhances the dependability of an untrusted system using a trusted monitor and recovery systems.

Run-time Assurance (RTA) techniques are intended to replace costly detailed design time risk analysis and static FDIR mechanisms (Schierman et al., 2015). They can improve the dependability of untrusted complex systems prioritizing performance over reliability, e.g., neural networks. Fig. 3 shows an RTA architecture applied to an Untrusted System that controls a Plant. The architecture involves a Trusted Monitor, a Trusted Recovery System and a Switch. The Trusted Monitor monitors the states of the Untrusted System and the Plant to detect early violations of a set of essential system properties. We assume that the Untrusted System and the Plant are instrumented to enhance the observability of faulty state changes. When a fault is detected, the Switch is activated to replace the output of the Untrusted System with the output of the Trusted Recovery System. The latter is able to provide a minimal service so that the Plant could avoid non-trustworthy states.

The successful application of this principle raises some non-trivial technical problems. Special care should be taken for the development of the trusted components to ensure that they effectively achieve the intended goals of fault detection and mitigation. The Trusted Monitor could be synthesized from the formal specification of the properties characterizing system correctness. It should be able to detect early enough their violation so that it could be still possible for the Recovery System to control the Plant safely. This requires a deep understanding and analysis of the Plant dynamics to identify when the Trusted Recovery System can safely take over. Furthermore, it is important to be able to anticipate and counteract risks adequately.

Following the principle of RTA as discussed above, the self-adaptive system concept has been proposed (Cheng et al., 2009; Weyns, 2017, 2021) to allow the target system to autonomously adapt to the evolution of the environment at run-time, both internally and externally, while still fulfilling its safety goal. Changes could be caused by unwanted and unexpected failures in the system or events in its running environment that could lead to hazardous situations. In this context, adaptation means that system can dynamically re-configure and adjust itself to achieve its goals, despite the uncertain changes that occur during operation. Fig. 4 illustrates a conceptual model of a self-adaptive system, in which a management system continuously senses the status of the evolving system states and its environment in order to make appropriate adaptive decisions. At the same time, interfaces to human stakeholders are defined to provide human oversight of the evolving system.

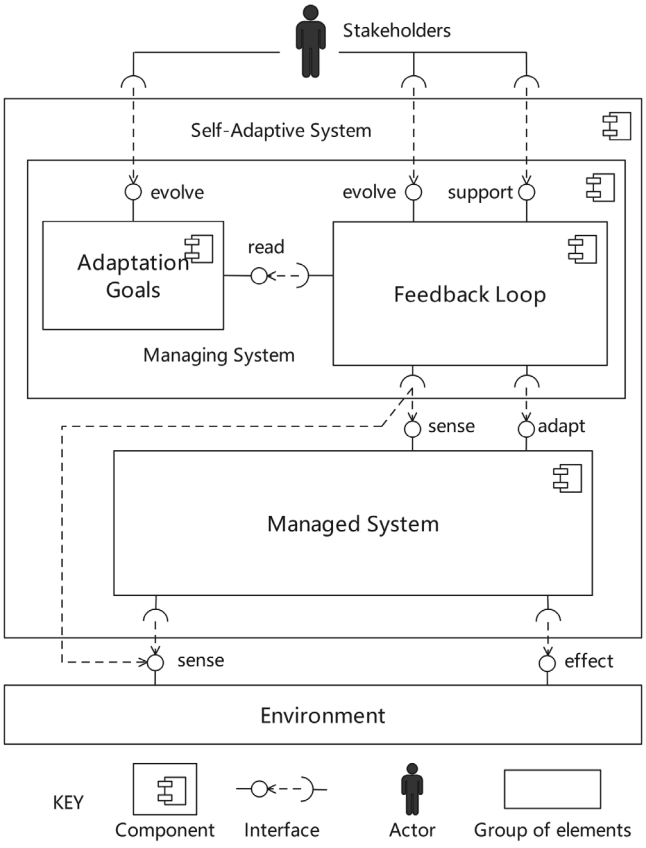

Fig. 4. A conceptual model of self-adaptive systems (Weyns, 2017).

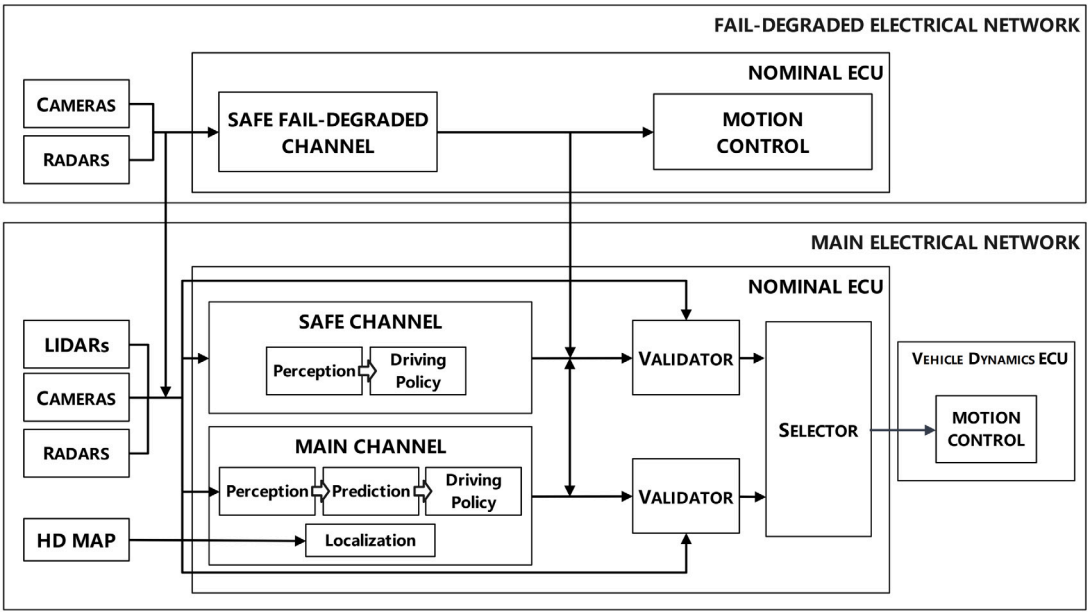

Fig. 5. BMW ADS redundancy concept (BMW group, 2020).

Within the management system, the adaptation goal expresses the concern to enforce properties on the managed system, typically related to optimization, healing, configuration and protection. The feedback mechanism is responsible for continuously monitoring the internal and external conditions, gaining situational awareness through analysis. Furthermore, it executes an adaptation strategy and plans actions on the managed system, based on the existing knowledge base. To support this architecture, methods such as safe run-time model, fuzzing, run-time verifications are potentially applied.

For autonomous driving systems, it is especially important to monitor the potential hazardous behavior of the system and to take swift action to mitigate the risk. Shalev-Shwartz et al. (2017) propose redundant architecture with a safety monitor. The latter uses the Responsibility Sensitive Safety (RSS) model, which provides the conditions for assessing the driving risk and taking concrete steps to avoid the collision. Online verification reachability analysis method with a fallback measure is also used to conduct the fall-safe strategy once the risk is detected (Althoff and Dolan, 2014; Pek et al., 2020).

The TTTech company proposes a fault-tolerant ADS architecture, which utilizes a run-time monitor to verify the latent failures and to decide whether to switch to the backup automated driving system in order to mitigate the incoming risk (Mehmed et al., 2020). Fig. 5 shows the structure of the BMW ADS implementing a redundancy mechanism, where two risk diagnostic units monitor a main channel and a safe channel and properly switch to a third degraded rudimentary channel in case of failure to mitigate risks (BMW group, 2020). The novelty of this structure lies in the use of two redundant channels to ensure the safety of the autonomous vehicle. The safe channel aims to maintain the basic driving task, and the degraded safe-failure channel is responsible for planning a fallback trajectory when the risk cannot be properly managed by the main channel.

3.3. Risk assessment

3.3.1. Risk assessment criteria

System dependability is specified by a set of attributes about the continuity, efficiency and quality of the service/mission for which the system is deployed (Avizienis et al., 2004). We consider below a few of them and show their impacts on both technical and business outcomes:

1. *Reliability R(t)* is the probability that the system works without failure up to time t. Reliability is important for systems where continuity of service matters, e.g., avionic systems, and less important for systems where service interrupts are not critical, e.g., mobile phones.
2. *Availability*, in its simplest form, is the measure of uptime (the amount of time the system is performing its mission) divided by the uptime plus downtime (the amount of time the system is not performing its mission). Hence, a system can be highly available even if it fails frequently while the recovery from failure is easy, e.g., a router can easily recover from data losses.
3. *Serviceability* measures the ease and speed of corrective maintenance and preventive maintenance that can be conducted on a system, e.g., in telecommunication systems where I/O modules can be easily replaced in case of failure.
4. *Manageability* is characterized by a set of functions of four broad categories: (1) *Health Monitoring, Logging, and Alerting* that keep track of the system's ability to perform as expected; (2) *Configuration and Control* that allow setting up and configuring the system correctly for optimal performance and availability; (3) *Deployment and Updates* allowing efficient and sometimes automatic deployment of system resources such as a software stack; (4) *Asset Discovery and Inventory* especially useful for networked or remote systems. This property is important for server farms where each server can be accessed and its software can be updated.

3.3.2. Analytical assessment techniques

Risk Assessment aims to estimate a variety of dependability attributes as introduced above. In this section, we will focus on the analytical techniques used for assessing the system reliability. Analytical techniques depict the behaviors of systems as mathematical models and estimate the reliability metrics using direct numerical solutions (Xie, 1991; Trivedi and Bobbio, 2017). Over the years, many reliability assessment methods have been proposed and each of them has its own application scenarios.

3.3.2.1. Reliability block diagram. Reliability Block Diagram (RBD) methods are used to estimate the reliability of the serial and parallel composition of components, with the assumption that the components are statistically independent (IEC61078, 2016; Catelani et al., 2019). If

R_1 and R_2 are the reliabilities of two components that work in tandem, the reliability of the system will be $R_1 R_2$. If the two components work in parallel, the reliability of the system will be $1-(1-R_1)(1-R_2)$ instead. Another basic composition scheme is the k-out-of-N system, for which the correct execution of the system requires that at least k components out of the N components execute correctly. The reliability of the system will be a binomial probability of $\sum_{i=k}^{N} \binom{i}{N} R^i (1-R)^{N-i}$.

Reliability assessment is conducted by firstly partitioning the system into appropriate logical blocks. Given the reliability of each block and the defined connections, the reliability of the whole system can be estimated using the basic rules mentioned above. One of the limitations of this method is that it can fail to model complex architectures, which generally cannot be expressed as a serial and parallel composition of components.

3.3.2.2. Reliability graph. Network abstraction is an effective way to deal with many complex systems that involve interconnections and complex relationships between elements/components. Reliability graphs are designed to analyze the reliability of networks (Colbourn, 1987). Many systems, such as computer systems, power grids, and telecommunication systems, are treated as multi-source multi-sink flow networks. The proper operation of such a network is based on the connection between every pair of source node (s) and sink node (t). The correct operation is thus characterized as the successful transmission of a given amount of data from s to t. When using a reliability graph to evaluate the network, network reliability is defined as the probability that at least one path with no failed edges and failed nodes from s to t exists. When the focus is a pair of source and sink, the assessment approach is known as the two-terminal reliability analysis. An all-terminal reliability is defined as the probability that all the pairs of source nodes and sink nodes can communicate.

After a reliability graph is built, and all the edges and nodes are assigned with reliabilities, the reliability of the modeled network can be estimated using some classical algorithms like the inclusion-exclusion formula (Dohmen, 1998), the sum of disjoint products (Xing et al., 2012), and Binary Decision Diagrams (Bryant, 1986; Xing and Amari, 2015).

Both RBD and reliability graph are categorized into stateless (combinational) models which involve the often unrealistic assumption that the components of the system behave in a statistically independent manner (Trivedi and Bobbio, 2017).

3.3.2.3. Markov chain. Markov chains are a well-known class of stochastic models used for reliability analysis. In contrast to the combinational models introduced above, Markov chains allow modeling statistical state and time dependencies (Xie, 1991; Trivedi and Bobbio, 2017). The behavior of a system is usually treated as a stochastic process that is represented as a sequence of discrete states. The execution sequence of states is modeled as a Discrete-time Markov Chain (DTMC) if the next state depends probabilistically on the present state but not on the past history (Cheung, 1980). For example, a sequence of component executions is often represented as a DTMC, and two terminal states S and F, which represent the successful completion and failure state of the system, respectively, are added to the DTMC. The DTMC is then characterized by its one-step transition probability matrix, $P = [p_{i,j}]$, where $p_{i,j}$ is the product of the reliability of component i and the transition probability between components i and j; this probability characterizes the control transfer from component i to component j, given the successful execution of component i. The reliability of the system is obtained by measuring the probability of the transition from the initial state to state S.

When system execution time is taken into account, the stochastic process is modeled as a Continuous-time Markov Chain (Laprie and Kanoun, 1992) which has an embedded DTMC with an exponential sojourn time distribution for each state. Several tools allow reliability analysis of Markov chains, e.g., SHARPE (Sahner et al., 1995), PCM (Brosch et al., 2012). This model suffers from limitations arising from the underlying assumptions, for example the Markov property, exponential sojourn times, only one component executing at a time (Gokhale, 2007; Trivedi and Bobbio, 2017). These assumptions are not always valid in practice. An additional limitation is state explosion, resulting in exponential computational complexity.

3.3.2.4. Petri net. The space explosion problem of the Markov chains can be alleviated by using expressive formalism that enables concise specification of the system behavior (Trivedi and Bobbio, 2017). Petri net (Petri, 1962) are such formalism from which a Markov chain representing their behavior can be automatically generated. Petri nets capture a system's dynamic behavior including concurrent, asynchronous, distributed, and parallel activities. Reliability analysis using Petri nets is based on calculating the probabilities of their steady states. The failure probability of the system is given by the probability of failure states. Petri nets have some variants that take into account times associated with transitions (stochastic Petri net or Time Petri net Ajmone Marsan et al., 1984; Ciardo et al., 1989), types of tokens (uncolored or colored Petri nets Jensen and Kristensen, 2009), and structural extensions including guard functions, variable cardinality arcs, and general marking dependency (Stochastic Reward Net Ciardo et al., 1993). These extensions can support the modeling and reliability assessment of systems of different characteristics. There are several tools available for modeling and analyzing Petri net and its variants (CPN IDE Homepage, 2024; TimeNet Homepage, 2018; Paolieri et al., 2021).

3.3.2.5. Bayesian network. Bayesian networks (BNs) are increasingly applied in reliability assessment (Langseth and Portinale, 2007; Chu et al., 2018; Cai et al., 2019), given their power of probabilistic knowledge representation and inference. They provide a flexible modeling framework to deal with various kinds of statistical dependences and to effectively characterize uncertainty by avoiding the state space explosion problem of Markov chains. Static BNs are directed acyclic graphs whose nodes represent variables and edges/arcs represent causal or influential dependencies between nodes with conditional probabilities qualifying the dependencies. BNs also have some variants that include temporal features (dynamic BNs) (Darwiche, 2009) and the concepts of class and object (object-oriented BNs) (Kjaerulff and Madsen, 2012). These variants can support the reliability assessment of dynamic systems with complex and hierarchical structures (Yuan et al., 2018).

Generally, the procedure of BN-based reliability assessment includes three steps:

1. The first step is to construct the BN models. The final leaf node usually represents the system reliability to be evaluated, and its parent nodes represent the reliabilities of components or other factors. Relevant knowledge regarding the nodes and cause-and-effect relationships is usually collected by experts and then incorporated into the BNs. Typically for hardware, the knowledge can be learned from reliability data (e.g., failure data of the system and its components) (Doguc and Emmanuel Ramirez-Marquez, 2012). Another way to build BN structure models is to generate them from other risk analysis models (e.g., Fault Tree; see Section 3.1.2.3) (Bobbio et al., 2001).
2. The second step is to build a node probability table (NPT), where the root nodes and others are allocated with prior and conditional distributions, respectively. The NPT can also be elicited from expert knowledge or learned from historical reliability data.
3. Subsequently, given new observations, the NPT are updated using various inference algorithms based on Bayes' theorem (Marquez et al., 2010; Zhong et al., 2010).

Numerous commercial tools, such as Hugin (Madsen et al., 2003) and Netica (Norsys Software Corp., 2023), are available for BN inference. Although exhibiting the greater modeling power than the other methods, the applications of BNs in reliability assessment is

limited. In practice, it is usually difficult to realize structure and parameter modeling based on expert knowledge, because the evaluated system may be too complex to model, and a large amount of expert knowledge is required. High subjectivity may also introduce high inaccuracy in reliability assessment. Nevertheless, improved Bayesian inference algorithms are still being investigated to reduce the current high computation cost of complex reliability models.

We have presented a brief overview of the mainstream analytical techniques for estimating reliability. Sometimes, to avoid state space explosion problems that are inherent to the modeling of complex, large-size, or hierarchical systems, an analytical hierarchical model that combines different techniques is used (Trivedi and Bobbio, 2017). This approach is typical for the reliability assessment of modern cloud data centers and virtual networks (Nguyen et al., 2019). Although the above discussions are in the context of reliability, these techniques are not specific to reliability assessment. Assessment of serviceability or availability, can also be realized on basis of these techniques, while recovery/repair behaviors of systems and the corresponding time consumed are taken into consideration.

3.3.3. Behavioral risk assessment techniques

Increasing complexity and dynamism of modern systems require more expressive analysis and assessment models and tools. Conventional risk analysis formalisms and methods express simple causality relations for the connection between risk causes and their manifestations. Most of them do not take into account states and system dynamics while use only models of limited expressiveness. As a result, the analysis and assessment lead to inaccurate worst-case estimation of risks (Epstein and Rauzy, 2005).

A key idea is to describe the system behavior in terms of states and transitions going beyond state/event formalisms such as Markov chains that suffer from the exponential explosion of states and transitions and the lack of structuring mechanisms. The focus is rather on faithful modeling showing all the relevant behavioral aspects concerning the satisfaction of essential system properties and the impact of hazards on this behavior. Taking such an approach limits the application of analytic techniques but connects to model-based systems engineering techniques and tools. The behavioral models used for dependability analysis are very close to the actual system models used in model-based design flows (Section 2).

For risk and safety assessment, general modeling formalisms have been used such as SysML (David et al., 2010), AADL (Stewart et al., 2021), or BIP (Nouri et al., 2018). Other more specific languages and supporting tools have been proposed such as Altarica 3.0 (Prosvirnova et al., 2013) dedicated to probabilistic risk and safety analyses of complex technical systems. The language allows the structured description of system behavior as the composition of hierarchically composed block diagrams. The tools allow diverse safety and reliability assessments, optimizations of maintenance policies, and assessments of the expected production level over a given period.

Model checking techniques have been extensively used for risk analysis and assessment. The NuSMV3 model checker has been used in a series of works (Bozzano et al., 2013) where it shows how model checking techniques can be used for reliability analysis of real-life applications. Statistical model checking has also been used for a timed/probabilistic extension of the BIP language (Nouri et al., 2018) to estimate the reliability of nontrivial systems. Finally, the PRISM probabilistic model checker (Kwiatkowska et al., 2011) has been used for the reliability analysis of systems, for example to analyze systems in the Safety Analysis Modeling Language used to specify Markov decision processes (Gudemann and Ortmeier, 2010).

The idea of behavioral analysis and assessment depicted in Fig. 6 is to build a model of the out-of-nominal system behavior B^{H} from a model of the nominal system behavior B, taking into account the results of risk analysis that identifies a list of hazards H and their probabilities. B^{H} has in addition to the states of B extra states that encode the presence of hazards and extra transitions showing the effects of hazards on system behavior. Some transitions can lead to states (remarked as light-gray filled circles) from which recovery is possible while other transitions may lead to fatal states (remarked as black filled circles) from which no recovery is possible and definitely violate some safety or security property.

Given B^{H}, probabilistic analysis techniques can be applied to estimate the probability that some property is violated. It is also possible to apply synthesis algorithms to compute strategies for avoiding bad states using recovery transitions.

3.3.4. Empirical validation techniques

3.3.4.1. Testing techniques. Model-based system development provides a common framework for controlling and coordinating development cycle activities. System models are necessary for verification but also for analysis and assessment purposes, e.g. to estimate system performance and reliability.

An important difference between verification and testing is that verification allows an in-depth understanding of the system behavior because it is applied to a system model and therefore there are no observability limitations implied by the system implementation. In contrast, testing consists in applying stimuli to system inputs and checking whether the observed behavior is the expected one. Thus, the type of properties that can be tested is limited to accessibility properties (testing that a certain operational condition can be reached when appropriate stimuli are applied). Typically, we cannot test the satisfaction of invariance properties, i.e., a condition that always holds, which are essential for system trustworthiness. This would require a full exploration of the state space of the system and is possible only through verification. Only violations of these invariance properties can be tested without any guarantee that they are satisfied.

Currently, empirical validation and assessment techniques are taking precedence over model-based techniques as several trends in intelligent systems are making model-based development obsolete. One trend is the overwhelming complexity of the systems that defeats our capability to develop system models. The other trend is the use of AI components that are not explainable.

These trends have already had a serious impact on engineering practices especially applied by the industry for the development of intelligent systems. To avoid the limitations inherent in model-based approaches, some high-tech companies are adopting end-to-end AI-based solutions, e.g., the work of Bojarski et al. (2016). Other industrial players seek hybrid designs integrating model-based and data-based components. Consequently, model-based analysis and assessment will play a limited role and the focus will be on testing real systems or simulated prototypes.

A test is a controlled experiment to find potential defects in a system, revealing the violation of a property P. Testing environments involve a System Under Test (SUT), a Test Case Generator (TCG), and an Oracle (O), as shown in Fig. 7. SUT has predefined inputs (controllable variables) and outputs (observable variables). TCG generates test sequences used to drive the execution of SUT, which generates the corresponding runs analyzed by O with respect to the property P characterizing the system's I/O correct behavior.

To validate the property P, the Oracle computes a function based on reasoning or empirical evidence, delivering a verdict, pass or fail. The function can be a machine-calculated algorithm or a human expert able to judge according to unambiguous and justifiable criteria.

The Oracle can compute for the tested property P, a score that characterize the extent to which it can be satisfied. The score can be average rate of success with a level and an interval of confidence based on statistical techniques (Denise et al., 2004; Gouraud et al., 2001). Based on the calculated score, the Oracle can guide the test case generator to adapt by selecting test cases that improve the efficiency of the testing process (Harman et al., 2015).

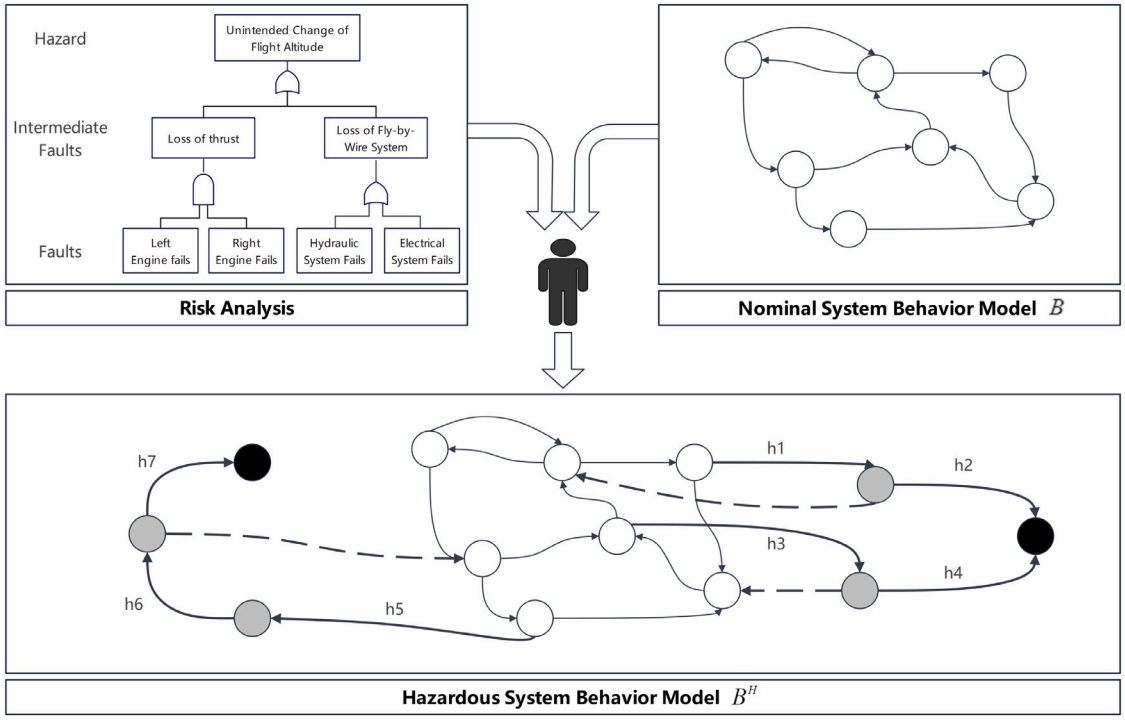

Fig. 6. Getting the hazardous system behavior.

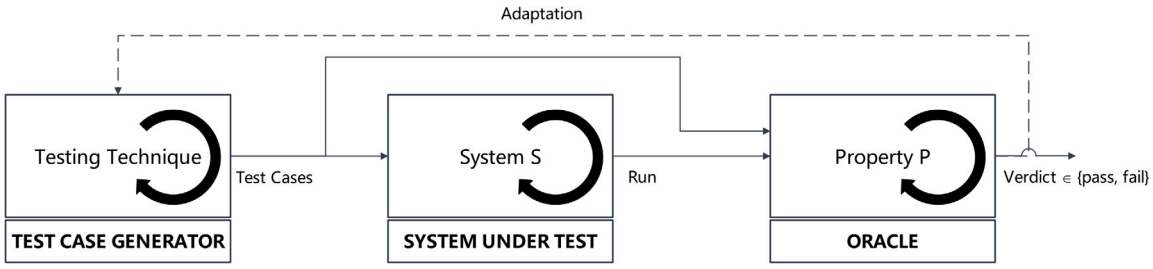

Fig. 7. General testing principle.

We often distinguish between black-box and white-box testing (Nidhra and Dondeti, 2012). For black-box testing, we only know the inputs and outputs of the system, whereas for white-box testing, we also have a model of the system's behavior. In the latter case, the testing process can be guided by the results of the system model analysis. White-box testing can be applied to test the source code of a given software or to a VLSI circuit specified in a hardware description language. However, for machine learning systems only black-box testing is possible.

A test campaign for a given SUT and a property P, is defined by a set of test cases TC that provide a sufficiently "good coverage" of P. Coverage criteria define to what extent the behavior of SUT is explored. They are mainly of two types: (1) Structural criteria that provide the percentage of the system structure exercised by the test cases, e.g., percentage of lines of source code, or percentage of branches of a control or data-flow graph. (2) Functional criteria that confirm that the system can perform some essential functions, e.g., deliver messages, break, adapt to stimuli, etc. For monolithic HW and SW systems it is possible to define coverage criteria based on their structure models (Wegener et al., 2001). For distributed systems, such as communication protocols, functional testing is more appropriate (Jard and Jéron, 2005).

How to select the test cases TC among all the possible stimuli? To cope with the complexity of the task, black-box testing techniques use an equivalence relation on test cases: two test cases tc1, tc2 are equivalent for the property P if the corresponding runs cannot be distinguished by the oracle O (Lee et al., 2020), i.e., either both runs satisfy P or they do not. We can thus reduce the testing complexity by considering only one test case per equivalence class. Note however that the existence of adversarial examples for neural networks does not allow this simplification: an adversarial example is a corrupted version of an input that is misclassified by the system while it cannot be distinguished by the Oracle.

When it comes to testing AI-enabled systems, several challenges arise because of their lack of explainability and their lack of robustness manifested by adversarial examples. One of the primary issues is generating suitable test cases that can adequately evaluate the system's performance. Test case generators for AI-enabled systems often randomly pick samples from a dataset of typical scenarios, including some challenging datasets that include previously failed cases or cases that failed in other AI models, such as ImageNet-A for ImageNet (Hendrycks et al., 2021). However, evaluating the system's robustness (Szegedy et al., 2013), fairness, and security requires more advanced techniques. Adversarial attacks are such a technique, and there are several practical tools available, such as Google Cleverhans (Papernot et al., 2016) and Bethgelab Foolbox (Rauber et al., 2017) which can generate adversarial examples. Attacks can be categorized into four fundamental types: gradient-based, score-based, decision-based, and transfer-based attacks. Some well-known attacks include FGSM (Goodfellow et al., 2014), C&W attack (Carlini and Wagner, 2017), and Local Search

Attack (Narodytska and Kasiviswanathan, 2016). ImageNet-C and ImageNet-P are two benchmark datasets for the assessment of model robustness against common corruptions and perturbations (Hendrycks and Dietterich, 2019).

Another significant problem when testing AI systems is determining an appropriate Oracle to compare the output of the AI system with the expected output. The Oracle problem is particularly challenging for AI systems that use deep learning or other complex algorithms, where it may be challenging to define what constitutes the expected output. Using Metamorphic Relations (Chen et al., 2018) and cross-referencing (Srisakaokul et al., 2018) as test Oracles are the two main methods. Metamorphic Relations is a technique that involves identifying input–output relationships that should remain consistent, irrespective of the changes made to the input. Cross-referencing, on the other hand, involves comparing the output generated by the AI system with the output generated by a human expert or another system, such as a well-established model. These methods can help define an appropriate Oracle for complex AI systems, making it easier to test and evaluate the system's performance.

An interesting question in the field of AI testing is how to measure testing coverage, which differs from traditional software development because the rules of an AI system are learned from data and unknown to the developers. Some researchers have tackled this issue. For example, Pei et al. (2017) proposed the first coverage criterion called neuron coverage, designed specifically for testing deep learning. Ma et al. (2018) extended this concept by profiling a DNN based on training data to obtain the activation behavior of each neuron. Sun et al. (2018) proposed four test coverage criteria tailored to the unique features of DNNs, inspired by MC/DC coverage criteria. Kim et al. (2019) introduced surprise adequacy to measure coverage of discretized input surprise range for deep learning systems. However, some researchers (e.g., Li et al. (2019)) have noted the limitations of structural coverage criteria for deep networks, as experiments with natural inputs showed no strong correlation between the number of misclassified inputs and structural coverage. The black-box nature of machine learning systems makes it unclear how such criteria directly relate to the system's decision logic. From the perspective of functional criteria, the intended function of an AI system is to work properly in different scenarios.

Validation of complex autonomous systems, in particular self-driving systems, by simulation and testing raises some very challenging issues. For example, claiming that an autonomous driving system is safe enough because it has driven 10 billion kilometers in simulation does not necessarily imply that the real system is safe (WAYMO, 2018). The simulated miles must be related to the "real miles" to show that the simulation deals fairly with the many different situations, e.g., different road types, traffic conditions, weather conditions, etc.

Note that for autonomous driving systems, it is possible to formalize the properties and associated validation techniques applied by the Oracle. For example, if the property is a traffic rule, it is possible to formally check whether the observed behavior violates this property (Bozga and Sifakis, 2021). Nevertheless, coverage criteria and theoretical estimation of test campaign scores are missing. The key issue is how we can choose sets of driving scenarios satisfying given coverage criteria. For this, we need to develop scenario sampling techniques based on a statistical analysis of models representing traffic patterns.

3.3.4.2. Fault injection techniques. Fault injection is a powerful validation technique for assessing system dependability. It realizes controlled experiments that accelerate the occurrence and propagation processes of failures, mainly for the following purposes: (1) identify dependability bottlenecks; (2) understand system behavior in the presence of faults; (3) determine the coverage of fault detection and recovery mechanisms; (4) evaluate the effectiveness of fault tolerance mechanisms. Generally speaking, the categorization of fault injection techniques could be considered from three perspectives corresponding to three questions: (1) what to inject; (2) when to inject; and (3) where to inject.

For a fault injection process, a failure usually represents a risk cause (defect) that may be activated. An active Failure may result in an invalid state of the system, namely Error. An Error may cause further errors within the system boundary, or it propagates outside the system boundary and be observed. Once the Error is visible outside the system boundary, a Fault occurs. The cycle Failure-Error-Fault represents a chain of threats for system dependability. In practice, failures or errors can be injected. Failures can be physical or software-oriented. Common types of physical failures include voltage, clock, electromagnetic, and optical ones. Errors are injected by changing system states. Some works (Ziade et al., 2004; Salih et al., 2022) have surveyed different types of failures and errors targeting both hardware and software systems.

While failures and errors conceptually answer the question of "what to inject" in the fault injection domain, the question of "when to inject" provides another view to categorize fault injection techniques. For different stages of system development, fault injection can be carried out on different targets. Fault injection over the actual system is common and plays a significant role in identifying problems in a complex system, e.g., Amazon Cloud. However, at a late stage in the system's lifecycle, this can lead to major losses if something gets out of hand. Rather than affecting the real system, fault injection on the simulated/emulated system involves no risk and allows for a sufficiently large number of fault injection methods. A simulated system is a software program that models the target system, and an emulated system is often implemented by Field Programmable Gate Arrays (FPGA) to prototype the target system. Although there is always a gap between the real system and the simulated/emulated one, it is possible to get valuable information of the system under fault injection with little risk. Simulation-based and emulation-based fault injection techniques (Kooli and Di Natale, 2014; Eslami et al., 2020) come in the relatively early stages and have dominated almost half of the research domain of fault injection. Moreover, in model-based systems engineering, system models come into play even earlier than simulated/emulated systems. Model-based fault injection consists of injecting faults into executable system models, in order to identify the effects of faults at a very early stage of development (Yoneyama et al., 2019; Fabarisov et al., 2021). On the other hand, fault injection techniques can be used to generate test cases and contribute to system analysis at later stages (Marrone et al., 2014).

From the perspective of "where to inject", fault injection could be over the software, the hardware, or the model of the target system. Fault injection over software could be done at compile-time and run-time (Hsueh et al., 1997; Natella et al., 2016). Three types of software-based fault injection approaches, i.e., data error injection, interface error injection and code change injection, are carefully surveyed in Natella et al. (2016). As an example, fuzzing or fuzz testing (Li et al., 2018; Liang et al., 2018) is an automated software testing technique that relies on providing designed software inputs, e.g., random or unexpected input data. With carefully designed inputs by a fuzzier, one could try to maximize the testing coverage of a program and identify possible defects of a program. Fault injection over hardware, i.e., hardware-based fault injection, on the other hand, can be divided into physical fault injection and logical fault injection (Entrena et al., 2011). For example, particle radiation or laser beam can be used to trigger faults, e.g., bit-flip, so as to evaluate the robustness of a circuit against environmental disturbances. Fault injection can be applied either to the actual software/hardware system, or to a model of it. Fig. 8 compares the two approaches. Fig. 8(a) shows a typical fault injection system (Hsueh et al., 1997) which generates workloads, injects faults into the real software/hardware system, and then monitors the system's operational data for analysis. Model-based fault injection uses an executable model of the system and its environment into which faults are injected. Fig. 8(b) shows fault injection in a system model, which can be analyzed to assess the effects of the injected faults.

If we examine the current state of the art in the field of fault injection, two striking trends emerge:

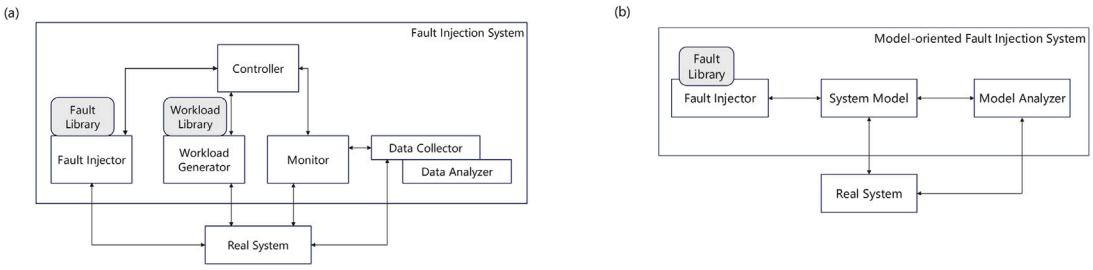

Fig. 8. (a) typical fault injection technique system (Hsueh et al., 1997) and (b) model-oriented fault injection system.

1. The last decade has seen a growing interest in the application of formal methods to fault injection and the analysis of fault tolerance mechanisms (Jayakumar, 2020). Methods such as model checking, assertion-based verification and symbolic analysis for fault injection are relatively new in this field. Fault injection in system models based on formal methods could develop rapidly in the coming years, given the widespread application of complex safety-critical systems, such as autonomous vehicles, which require validation in the early phases of system development. However, its effective application requires modeling languages rooted in operational semantics and capable of faithfully describing the dynamics of the system's interaction with its environment, as well as the effect of faults.
2. Not surprisingly, another growing trend is fault injection over AI systems because of their proliferation and the fact that the AI technical stack is distinct from the traditional hardware and software stacks. The AI technical stack has specific hardware, e.g. a GPU or NPU, on which AI operators are implemented to boost performance of machines learning platforms, e.g., Tensorflow or MindSpore. Some works (Tsai et al., 2021; Guerrero-Balaguera et al., 2022) inject faults over GPU/NPU, which are the underlying processing units for many commercial AI models/applications. Other works (Chen et al., 2019; Zheng et al., 2021) target the fault injection over AI/Machine Learning software libraries or frameworks. In addition, the growing demand for safe AI has stimulated research into fault injection in AI models, e.g. deep neural networks (Liu et al., 2017), which is set to develop further.

4. Conclusion and perspectives

Currently, economic and technological factors in favor of intelligent and autonomous systems are driving new trends that contradict the well-established practice of dependable systems engineering:

- not following the concept of "security by design" and instead adopting end-to-end solutions, based on machine learning, which cannot provide sufficient trustworthiness guarantees;
- allowing "self-certification", in the absence of standards, that is the manufacturer guarantees system trustworthiness and not an independent authority;
- allowing regular updates of critical software, which implies that trustworthiness cannot be guaranteed at design time as required by standards and that systems will be evolvable, with no end point in their evolution.
- Increasing demands for dynamic treatments of run-time residual risks posed by open world interactions between autonomous systems and their changing environment, as the boundary of safety assurance between design and run-time blurs.

These trends will have definitely an impact on dependability techniques. The role of verification will gradually diminish, being applied only to components, while empirical techniques based on simulation and testing will dominate the systems validation landscape. This evolution will have profound consequences on the level of certainty and the degree of evidence provided for system dependability. Obviously, it will not be possible to guarantee at design time bounds for failure rates, such as the famous 10^{-9} number given for the maximum probability of a catastrophic failure, per hour of operation, in life-critical systems such as commercial aircraft (Aviation, 2017).

The important question is to what extent it will be possible to develop systems of guaranteed dependability despite their complexity and the integration of AI components.

To close the gap between the current state of the art and the emerging needs, we need new foundations for systems engineering allowing the integration of traditional model-based approaches and data-driven AI techniques to take the best from each.

One step in that direction would be the development of explainable AI techniques allowing extraction from AI components of their corresponding behavioral model. This is theoretically possible for feed-forward neural networks, given their structure and the mathematical function that characterizes the input/output behavior of their nodes (Katz et al., 2017; Franco et al., 2022). Computing adequate abstractions of these functions could pave the way for formal verification as well as for sensitivity analysis. This effort should be complemented by work to identify the causes of adversarial examples and address the resulting anomalies.

At the same time, systems engineering should respond to current needs by developing foundations to compensate for the loss of predictability of model-based techniques by working in two directions.

The first is the development of correct-by-design solutions using reconfigurable architectures that can support hybrid designs such as those adopted by run-time assurance techniques. Instead of using passive redundancy that works well for hardware systems, hybrid designs will incorporate untrusted intelligent components and trusted model-based monitors that can take over in critical situations. Hybrid solutions will play an important role in future intelligent systems. They will combine traditional techniques to build trusted components for property monitoring and enforcement with reconfiguration techniques allowing self-organization of the system. In hybrid architectures, components should act in synergy to achieve resilience despite dynamically changing goals and environmental conditions. For example, self-healing (Schneider et al., 2015) requires successively (1) the detection of risks by system monitors; (2) the mitigation of risks to allow a minimal availability of the system; (3) the recovery of the system through self-organization.

The second direction is to develop a testing theory for intelligent systems to assess the likelihood that they will satisfy critical properties. We need statistics-based techniques to estimate the degree of coverage of given properties. This problem is closely related to explainability, since the existence of models can serve as a basis for developing coverage criteria.

In conclusion, the marriage with AI can be an excellent opportunity to reinvigorate systems engineering and, why not, tame AI techniques to develop even more dependable systems.

CRediT authorship contribution statement

Hezhen Liu: Conceptualization, Investigation, Supervision, Writing – original draft, Writing – review & editing. **Chengqiang Huang:** Conceptualization, Supervision, Writing – original draft, Writing – review

& editing. **Ke Sun:** Investigation, Writing – original draft, Writing – review & editing. **Jiacheng Yin:** Investigation, Writing – original draft, Writing – review & editing. **Xiaoyu Wu:** Investigation, Writing – original draft, Writing – review & editing. **Jin Wang:** Investigation, Writing – original draft, Writing – review & editing. **Qunli Zhang:** Investigation, Writing – original draft, Writing – review & editing. **Yang Zheng:** Investigation, Writing – original draft, Writing – review & editing. **Vivek Nigam:** Investigation, Writing – original draft, Writing – review & editing. **Feng Liu:** Investigation, Writing – original draft, Writing – review & editing. **Joseph Sifakis:** Conceptualization, Project administration, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

Ajmone Marsan, M., Conte, G., Balbo, G., 1984. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. ACM Trans. Comput. Syst. 2 (2), 93–122. http://dx.doi.org/10.1145/190.191.

Althoff, M., Dolan, J.M., 2014. Online verification of automated road vehicles using reachability analysis. IEEE Trans. Robot. 30 (4), 903–918. http://dx.doi.org/10.1109/TRO.2014.2312453.

Amin, Z., Singh, H., Sethi, N., 2015. Review on fault tolerance techniques in cloud computing. Int. J. Comput. Appl. 116 (18), 11–17.

Apostolakis, G., 2004. How useful is quantitative risk assessment? Risk Anal. 24, 515–520. http://dx.doi.org/10.1111/j.0272-4332.2004.00455.x.

Ashwin, T., Barai, A., Uddin, K., Somerville, L., McGordon, A., Marco, J., 2018. Prediction of battery storage ageing and solid electrolyte interphase property estimation using an electrochemical model. J. Power Sources 385, 141–147.

Aviation, 2017. What are the design parameters for airliner safety? https://aviation.stackexchange.com/questions/46677/what-are-the-design-parameters-for-airliner-safety.

Avizienis, A., 1985. The N-version approach to fault-tolerant software. IEEE Trans. Softw. Eng. SE-11 (12), 1491–1501. http://dx.doi.org/10.1109/TSE.1985.231893.

Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C., 2004. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secure Comput. 1 (1), 11–33. http://dx.doi.org/10.1109/TDSC.2004.2.

Bittner, B., Bozzano, M., Cimatti, A., De Ferluc, R., Gario, M., Guiotto, A., Yushtein, Y., 2014. An integrated process for FDIR design in aerospace. In: Ortmeier, F., Rauzy, A. (Eds.), Model-Based Safety and Assessment. Springer International Publishing, Cham, pp. 82–95.

BMW group, 2020. Safety assessment report: SAE level 3 automated driving system. https://usermanual.wiki/m/4bc317041f2a935a1043a71c6f17e878c4b17dddbbf798f19c27a4e0be22bfbb.pdf.

Bobbio, A., Portinale, L., Minichino, M., Ciancamerla, E., 2001. Improving the analysis of dependable systems by mapping fault trees into Bayesian networks. Reliab. Eng. Syst. Saf. 71 (3), 249–260. http://dx.doi.org/10.1016/S0951-8320(00)00077-6.

Bojarski, M., Firner, B., Flepp, B., Jackel, L., Muller, U., Zieba, K., Testa, D.D., 2016. End-to-end deep learning for self-driving cars. https://developer.nvidia.com/blog/deep-learning-self-driving-cars/.

Bozga, M., Sifakis, J., 2021. Specification and validation of autonomous driving systems: A multilevel semantic framework. CoRR abs/2109.06478. arXiv:2109.06478.

Bozzano, M., Cimatti, A., Mattarei, C., 2013. Automated analysis of reliability architectures. In: 2013 18th International Conference on Engineering of Complex Computer Systems. pp. 198–207. http://dx.doi.org/10.1109/ICECCS.2013.37.

Brosch, F., Koziolek, H., Buhnova, B., Reussner, R., 2012. Architecture-based reliability prediction with the palladio component model. IEEE Trans. Softw. Eng. 38 (6), 1319–1339. http://dx.doi.org/10.1109/TSE.2011.94.

Bryant, 1986. Graph-based algorithms for boolean function manipulation. IEEE Trans. Comput. C-35 (8), 677–691. http://dx.doi.org/10.1109/TC.1986.1676819.

Cai, B., Kong, X., Liu, Y., Lin, J., Yuan, X., Xu, H., Ji, R., 2019. Application of Bayesian networks in reliability evaluation. IEEE Trans. Ind. Inform. 15 (4), 2146–2157. http://dx.doi.org/10.1109/TII.2018.2858281.

Carlini, N., Wagner, D., 2017. Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy. SP, pp. 39–57.

Catelani, M., Ciani, L., Venzi, M., 2019. RBD model-based approach for reliability assessment in complex systems. IEEE Syst. J. 13 (3), 2089–2097. http://dx.doi.org/10.1109/JSYST.2018.2840220.

Čepin, M., 2011. Assessment of Power System Reliability: Methods and Applications. Springer Science & Business Media.

Chawla, S., Gionis, A., 2013. k-means–: A unified approach to clustering and outlier detection. In: Proceedings of the 2013 SIAM International Conference on Data Mining. SIAM, pp. 189–197.

Chen, T.Y., Kuo, F.-C., Liu, H., Poon, P.-L., Towey, D., Tse, T., Zhou, Z.Q., 2018. Metamorphic testing: A review of challenges and opportunities. ACM Comput. Surv. 51 (1), 1–27.

Chen, Z., Li, G., Pattabiraman, K., DeBardeleben, N., 2019. Binfi: An efficient fault injector for safety-critical machine learning systems. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC '19, Association for Computing Machinery, http://dx.doi.org/10.1145/3295500.3356177.

Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Di Marzo Serugendo, G., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H.M., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Müller, H.A., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., Whittle, J., 2009. Software engineering for self-adaptive systems: A research roadmap. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (Eds.), Software Engineering for Self-Adaptive Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1–26. http://dx.doi.org/10.1007/978-3-642-02161-9_1.

Cheng, Z., Zou, C., Dong, J., 2019. Outlier detection using isolation forest and local outlier factor. In: Proceedings of the Conference on Research in Adaptive and Convergent Systems. pp. 161–168.

Cheraghlou, M.N., Khadem-Zadeh, A., Haghparast, M., 2016. A survey of fault tolerance architecture in cloud computing. J. Netw. Comput. Appl. 61, 81–92.

Cheung, R., 1980. A user-oriented software reliability model. IEEE Trans. Softw. Eng. SE-6 (2), 118–125. http://dx.doi.org/10.1109/TSE.1980.234477.

Chu, T.-L., Varuttamaseni, A., Yue, M., Lee, S.J., Kang, H.G., Cho, J., Yang, S., 2018. Developing a Bayesian Belief Network Model for Quantifying the Probability of Software Failure of a Protection System. Vol. NUREG/CR-7233, United State Nuclear Regulatory Commission.

Ciardo, G., Blakemore, A., Chimento, P.F., Muppala, J.K., Trivedi, K.S., 1993. Automated generation and analysis of Markov reward models using stochastic reward nets. In: Meyer, C.D., Plemmons, R.J. (Eds.), Linear Algebra, Markov Chains, and Queueing Models. Springer New York, New York, NY, pp. 145–191.

Ciardo, G., Muppala, J.K., Trivedi, K.S., 1989. SPNP: stochastic Petri net package. In: Proceedings of the Third International Workshop on Petri Nets and Performance Models. PNPM89, pp. 142–151.

Colbourn, C.J., 1987. The Combinatorics of Network Reliability. Oxford University Press, Inc..

Coppens, B., De Sutter, B., Volckaert, S., 2018. Multi-variant execution environments. In: The Continuing Arms Race: Code-Reuse Attacks and Defenses. pp. 211–258.

Costa, C.H., Park, Y., Rosenburg, B.S., Cher, C.-Y., Ryu, K.D., 2014. A system software approach to proactive memory-error avoidance. In: SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 707–718. http://dx.doi.org/10.1109/SC.2014.63.

CPN IDE homepage, 2024. https://cpnide.org/.

Darwiche, A., 2009. Modeling and Reasoning with Bayesian Networks. Cambridge University Press, http://dx.doi.org/10.1017/CBO9780511811357.

David, P., Idasiak, V., Kratz, F., 2010. Reliability study of complex physical systems using SysML. Reliab. Eng. Syst. Saf. 95 (4), 431–450. http://dx.doi.org/10.1016/j.ress.2009.11.015.

Day, J.C., Donahue, K., Ingham, M.D., Kadesch, A., Kennedy, A., Post, E., 2012. Modeling Off-Nominal Behavior in SysML. Infotech@Aerospace.

Delange, J., Feiler, P., Gluch, D.P., Hudak, J., 2014. AADL Fault Modeling and Analysis Within an ARP4761 Safety Assessment. Carnegie Mellon University.

Denise, A., Gaudel, M.-C., Gouraud, S.-D., 2004. A generic method for statistical testing. In: 15th International Symposium on Software Reliability Engineering. pp. 25–34. http://dx.doi.org/10.1109/ISSRE.2004.2.

Doguc, O., Emmanuel Ramirez-Marquez, J., 2012. An automated method for estimating reliability of grid systems using Bayesian networks. Reliab. Eng. Syst. Saf. 104, 96–105. http://dx.doi.org/10.1016/j.ress.2012.03.016.

Dohmen, K., 1998. Inclusion-exclusion and network reliability. Electron. J. Combin. R36–R36.

Entrena, L., López-Ongil, C., García-Valderas, M., Portela-García, M., Nicolaidis, M., 2011. Hardware Fault Injection. Springer, pp. 141–166.

Epstein, S., Rauzy, A., 2005. Can we trust pra? Reliab. Eng. Syst. Saf. 88 (3), 195–205. http://dx.doi.org/10.1016/j.ress.2004.07.013.

Eslami, M., Ghavami, B., Raji, M., Mahani, A., 2020. A survey on fault injection methods of digital integrated circuits. Integration 71, 154–163. http://dx.doi.org/10.1016/j.vlsi.2019.11.006.

Fabarisov, T., Mamaev, I., Morozov, A., Janschek, K., 2021. Model-based fault injection experiments for the safety analysis of exoskeleton system. arXiv preprint arXiv:2101.01283.

Fenelon, P., McDermid, J.A., 1993. An integrated tool set for software safety analysis. J. Syst. Softw. 21 (3), 279–290.

Franco, N., Wollschlaeger, T., Gao, N., Lorenz, J.M., Guennemann, S., 2022. Quantum robustness verification: A hybrid quantum-classical neural network certification algorithm. arXiv:2205.00900.

Gokhale, S.S., 2007. Architecture-based software reliability analysis: Overview and limitations. IEEE Trans. Dependable Secure Comput. 4 (1), 32–40. http://dx.doi.org/10.1109/TDSC.2007.4.

Goodfellow, I.J., Shlens, J., Szegedy, C., 2014. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572.

Gouraud, S.-D., Denise, A., Gaudel, M.-C., Marre, B., 2001. A new way of automating statistical testing methods. In: Proceedings 16th Annual International Conference on Automated Software Engineering. ASE 2001, pp. 5–12. http://dx.doi.org/10.1109/ASE.2001.989785.

Gudemann, M., Ortmeier, F., 2010. A framework for qualitative and quantitative formal model-based safety analysis. In: 2010 IEEE 12th International Symposium on High Assurance Systems Engineering. pp. 132–141. http://dx.doi.org/10.1109/HASE.2010.24.

Guerrero-Balaguera, J.D., Galasso, L., Sierra, R.L., Reorda, M.S., 2022. Reliability assessment of neural networks in gpus: A framework for permanent faults injections. In: 2022 IEEE 31st International Symposium on Industrial Electronics. ISIE, IEEE, pp. 959–962.

Gui, H., Xu, Y., Bhasin, A., Han, J., 2015. Network a/b testing: From sampling to estimation. In: Proceedings of the 24th International Conference on World Wide Web. pp. 399–409.

Hamming, R.W., 1950. Error detecting and error correcting codes. Bell Syst. Tech. J. 29 (2), 147–160. http://dx.doi.org/10.1002/j.1538-7305.1950.tb00463.x.

Harman, M., Jia, Y., Zhang, Y., 2015. Achievements, open problems and challenges for search based software testing. In: 2015 IEEE 8th International Conference on Software Testing, Verification and Validation. ICST, pp. 1–12. http://dx.doi.org/10.1109/ICST.2015.7102580.

Hendrycks, D., Dietterich, T., 2019. Benchmarking neural network robustness to common corruptions and perturbations. arXiv preprint arXiv:1903.12261.

Hendrycks, D., Zhao, K., Basart, S., Steinhardt, J., Song, D., 2021. Natural adversarial examples. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 15262–15271.

Herder, J.N., Bos, H., Gras, B., Homburg, P., Tanenbaum, A.S., 2009. Fault isolation for device drivers. In: 2009 IEEE/IFIP International Conference on Dependable Systems & Networks. pp. 33–42. http://dx.doi.org/10.1109/DSN.2009.5270357.

Hsueh, M.-C., Tsai, T., Iyer, R., 1997. Fault injection techniques and tools. Computer 30 (4), 75–82. http://dx.doi.org/10.1109/2.585157.

Hu, W., Gao, J., Li, B., Wu, O., Du, J., Maybank, S., 2018. Anomaly detection using local kernel density estimation and context-based regression. IEEE Trans. Knowl. Data Eng. 32 (2), 218–233.

Hwang, I., Kim, S., Kim, Y., Seah, C.E., 2010. A survey of fault detection, isolation, and reconfiguration methods. IEEE Trans. Control Syst. Technol. 18 (3), 636–653. http://dx.doi.org/10.1109/TCST.2009.2026285.

IEC61078, 2016. Reliability Block Diagrams. IEC Standard.

ISO 21448:2022, 2022. Road Vehicles — Safety of the Intended Functionality. International Organization for Standardization, Vernier, Geneva.

ISO 26262-1:2018, 2018. Road Vehicles — Functional Safety. International Organization for Standardization, Vernier, Geneva.

ISO/IEC/IEEE 15288:2015, 2015. International standard - systems and software engineering – system life cycle processes.

Jard, C., Jéron, T., 2005. TGV: Theory, principles and algorithms: A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. Int. J. Softw. Tools Technol. Transfer 7 (4), 297–315.

Jayakumar, A.V., 2020. Systematic Model-Based Design Assurance and Property-Based Fault Injection for Safety Critical Digital Systems (Ph.D. thesis).

Jensen, K., Kristensen, L.M., 2009. Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer Berlin, Heidelberg, http://dx.doi.org/10.1007/b95112.

Joy, A.M., 2015. Performance comparison between linux containers and virtual machines. In: 2015 International Conference on Advances in Computer Engineering and Applications. pp. 342–346. http://dx.doi.org/10.1109/ICACEA.2015.7164727.

Ju, W.-h., 2016. Study on fire risk and disaster reducing factors of cotton logistics warehouse based on event and fault tree analysis. Procedia Eng. 135, 418–426.

Kadri, N., Koudil, M., 2019. A survey on fault-tolerant application mapping techniques for network-on-chip. J. Syst. Archit. 92, 39–52.

Kaseb, M.R., et al., 2019. An improved technique for increasing availability in big data replication. Future Gener. Comput. Syst. 91, 493–505.

Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J., 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. CoRR abs/1702.01135. arXiv:1702.01135.

Kececioglu, D., 1991. Reliability Engineering Handbook (Vol. 1). Prentice-Hall, Inc., USA.

Kevin Forsberg, H.C., 2005. Visualizing Project Management, third ed. John Wiley & Sons, Ltd, New York, NY.

Kim, J., Feldt, R., Yoo, S., 2019. Guiding deep learning system testing using surprise adequacy. In: 2019 IEEE/ACM 41st International Conference on Software Engineering. ICSE, IEEE, pp. 1039–1049.

Kjaerulff, U.B., Madsen, A.L., 2012. Bayesian networks and influence diagrams: A guide to construction and analysis. http://dx.doi.org/10.1007/978-1-4614-5104-4.

Kooli, M., Di Natale, G., 2014. A survey on simulation-based fault injection tools for complex systems. In: 2014 9th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era. DTIS, IEEE.

Kumari, P., Kaur, P., 2021. A survey of fault tolerance in cloud computing. J. King Saud Univ. Comput. Inf. Sci. 33 (10), 1159–1176.

Kwiatkowska, M., Norman, G., Parker, D., 2011. PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (Eds.), Computer Aided Verification. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 585–591.

Langseth, H., Portinale, L., 2007. Bayesian networks in reliability. Reliab. Eng. Syst. Saf. 92 (1), 92–108. http://dx.doi.org/10.1016/j.ress.2005.11.037.

Laprie, J., Kanoun, K., 1992. X-Ware reliability and availability modeling. IEEE Trans. Softw. Eng. 18 (02), 130–147. http://dx.doi.org/10.1109/32.121755.

Le, V.-H., Zhang, H., 2022. Log-based anomaly detection with deep learning: How far are we? In: Proceedings of the 44th International Conference on Software Engineering. pp. 1356–1367.

Lee, B., 2001. Using Bayes belief networks in industrial FMEA modeling and analysis. pp. 7–15.

Lee, J., Kang, S., Jung, P., 2020. Test coverage criteria for software product line testing: Systematic literature review. Inf. Softw. Technol. 122, 106272. http://dx.doi.org/10.1016/j.infsof.2020.106272.

Leveson, N.G., 2019. Shortcomings of the bow tie and other safety tools based on linear causality.

Leveson, N.G., Thomas, J.P., 2018. STPA Handbook. Massachusetts Institute of Technology.

Levy, S., et al., 2020. Predictive and adaptive failure mitigation to avert production cloud VM interruptions. In: Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation. OSDI, USENIX Association.

Li, Z., Ma, X., Xu, C., Cao, C., 2019. Structural coverage criteria for neural networks could be misleading. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results. ICSE-NIER, IEEE, pp. 89–92.

Li, J., Zhao, B., Zhang, C., 2018. Fuzzing: a survey. Cybersecurity 1 (1), 1–13.

Liang, H., Pei, X., Jia, X., Shen, W., Zhang, J., 2018. Fuzzing: State of the art. IEEE Trans. Reliab. 67 (3), 1199–1218.

Liu, J., Qin, C., Yu, Y., 2020. Enhancing distribution system resilience with proactive islanding and RCS-based fast fault isolation and service restoration. IEEE Trans. Smart Grid 11 (3), 2381–2395. http://dx.doi.org/10.1109/TSG.2019.2953716.

Liu, Y., Wei, L., Luo, B., Xu, Q., 2017. Fault injection attack on deep neural network. In: 2017 IEEE/ACM International Conference on Computer-Aided Design. ICCAD, pp. 131–138. http://dx.doi.org/10.1109/ICCAD.2017.8203770.

Liu, T., et al., 2019. A fault-tolerant neural network architecture. In: Proceedings of the 56th Annual Design Automation Conference 2019.

Ma, L., Juefei-Xu, F., Zhang, F., Sun, J., Xue, M., Li, B., Chen, C., Su, T., Li, L., Liu, Y., et al., 2018. Deepgauge: Multi-granularity testing criteria for deep learning systems. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. pp. 120–131.

Ma, Z., Wang, S., Liao, H., Zhang, C., 2019. Engineering-driven performance degradation analysis of hydraulic piston pump based on the inverse Gaussian process. Qual. Reliab. Eng. Int. 35 (7), 2278–2296.

Madsen, A., Lang, M., Kjærulff, U., Jensen, F., 2003. The hugin tool for learning Bayesian networks. 2711, 594–605. http://dx.doi.org/10.1007/978-3-540-45062-7_49.

Markowski, A.S., Kotynia, A., 2011. "Bow-tie" model in layer of protection analysis. Process Saf. Environ. Protect. 89 (4), 205–213.

Marquez, D., Neil, M., Fenton, N., 2010. Improved reliability modeling using Bayesian networks and dynamic discretization. Reliab. Eng. Syst. Saf. 95 (4), 412–425. http://dx.doi.org/10.1016/j.ress.2009.11.012.

Marrone, S., Flammini, F., Mazzocca, N., Nardone, R., Vittorini, V., 2014. Towards model-driven v&v assessment of railway control systems. Int. J. Softw. Tools Technol. Transfer 16, 669–683.

Massey, J.L., García, O., 1992. Error-correcting codes in computer arithmetic. In: Advances in Information Systems Science. Springer, Boston, MA, http://dx.doi.org/10.1007/978-1-4615-9053-8_5.

Mehmed, A., Steiner, W., Čaušević, A., 2020. Systematic false positive mitigation in safe automated driving systems. In: 2020 International Symposium on Industrial Electronics and Applications. INDEL, pp. 1–8. http://dx.doi.org/10.1109/INDEL50386.2020.9266146.

Meynen, S., Hohmann, S., Feßler, D., 2020. Robust fault detection and isolation for distributed and decentralized systems. In: 2020 IEEE International Conference on Systems, Man, and Cybernetics. SMC, pp. 401–407. http://dx.doi.org/10.1109/SMC42975.2020.9282986.

Microsoft contributors, 2023. Fault domain awareness. https://learn.microsoft.com/en-us/windows-server/failover-clustering/fault-domains.

Mohammed, B., Kiran, M., Awan, I.-U., Maiyama, K.M., 2016. An integrated virtualized strategy for fault tolerance in cloud computing environment. In: 2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld). IEEE, pp. 542–549.

Müller, S., Mikaelyan, L., Gerndt, A., Noll, T., 2020. Synthesizing and optimizing FDIR recovery strategies from fault trees. Sci. Comput. Program. 196, 102478. http://dx.doi.org/10.1016/j.scico.2020.102478.

Muniyandi, A.P., Rajeswari, R., Rajaram, R., 2012. Network anomaly detection by cascading k-means clustering and C4. 5 decision tree algorithm. Procedia Eng. 30, 174–182.

Nagalingam, D., Quah, A., Moon, S., Parab, S., Ng, P., Ting, S., Ma, H., Chen, C., 2020. Enhancing die level static fault isolation on power gated devices. Microelectron. Reliab. 108, 113629.

Narodytska, N., Kasiviswanathan, S.P., 2016. Simple black-box adversarial perturbations for deep networks. arXiv preprint arXiv:1612.06299.

Natella, R., Cotroneo, D., Madeira, H.S., 2016. Assessing dependability with software fault injection: A survey. ACM Comput. Surv. 48 (3), 1–55.

Nazari Cheraghlou, M., Khademzadeh, A., Haghparast, M., 2019. New fuzzy-based fault tolerance evaluation framework for cloud computing. J. Netw. Syst. Manage. 27, 930–948.

von Neumann, J., 1956. Probabilistic logics and synthesis of reliable organisms from unreliable components. In: Shannon, C., McCarthy, J. (Eds.), Automata Studies. Princeton University Press, pp. 43–98.

Nguyen, T.A., Min, D., Choi, E., Tran, T.D., 2019. Reliability and availability evaluation for cloud data center networks using hierarchical models. IEEE Access 7, 9273–9313. http://dx.doi.org/10.1109/ACCESS.2019.2891282.

Nguyen, H.D., Tran, K.P., Thomassey, S., Hamad, M., 2021. Forecasting and anomaly detection approaches using LSTM and LSTM autoencoder techniques with the applications in supply chain management. Int. J. Inf. Manage. 57, 102282.

Nidhra, S., Dondeti, J., 2012. Black box and white box testing techniques - A literature review. Int. J. Embedded Syst. Appl. 2 (2), 29–50. http://dx.doi.org/10.5121/ijesa.2012.2204.

Norsys Software Corp., 2023. Netica 6.08 Bayesian network software from norsys. http://www.norsys.com.

Nouri, A., Mediouni, B.L., Bozga, M., Combaz, J., Bensalem, S., Legay, A., 2018. Performance evaluation of stochastic real-time systems with the SBIP framework. Int. J. Crit. Comput. Based Syst. 8, 340–370. http://dx.doi.org/10.1504/IJCCBS.2018.096439.

Paltrinieri, N., Tugnoli, A., Buston, J., Wardman, M., Cozzani, V., 2013. Dynamic procedure for atypical scenarios identification (DyPASI): a new systematic HAZID tool. J. Loss Prev. Process Ind. 26 (4), 683–695.

Paolieri, M., Biagi, M., Carnevali, L., Vicario, E., 2021. The ORIS tool: Quantitative evaluation of non-Markovian systems. IEEE Trans. Softw. Eng. 47 (6), 1211–1225. http://dx.doi.org/10.1109/TSE.2019.2917202.

Papernot, N., Faghri, F., Carlini, N., Goodfellow, I., Feinman, R., Kurakin, A., Xie, C., Sharma, Y., Brown, T., Roy, A., et al., 2016. Technical report on the cleverhans v2. 1.0 adversarial examples library. arXiv preprint arXiv:1610.00768.

Park, S., 2017. Data-Based Semi-Automatic Hazard Identification for More Comprehensive Identification of Hazardous Scenarios (Ph.D. thesis).

Patrick O'Connor, A.K., 2012. Practical Reliability Engineering, fifth ed. John Wiley & Sons, Ltd, Chichester, UK.

Peach, G., Pan, R., Wu, Z., Parmer, G., Haster, C., Cherkasova, L., 2020. eWASM: Practical software fault isolation for reliable embedded devices. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (11), 3492–3505. http://dx.doi.org/10.1109/TCAD.2020.3012647.

Pei, K., Cao, Y., Yang, J., Jana, S., 2017. Deepxplore: Automated whitebox testing of deep learning systems. In: Proceedings of the 26th Symposium on Operating Systems Principles. pp. 1–18.

Pek, C., Manzinger, S., Koschi, M., Althoff, M., 2020. Using online verification to prevent autonomous vehicles from causing accidents. Nat. Mach. Intell. 2, 518–528.

Petri, C., 1962. Kommunikation mit Automaten (Doctoral Thesis). University of Bonn, West Germany.

Price, C.J., Taylor, N.S., 2002. Automated multiple failure FMEA. Reliab. Eng. Syst. Saf. 76 (1), 1–10.

Proskurin, S., Momeu, M., Ghavamnia, S., Kemerlis, V.P., Polychronakis, M., 2020. xMP: Selective memory protection for kernel and user space. In: 2020 IEEE Symposium on Security and Privacy. SP, pp. 563–577. http://dx.doi.org/10.1109/SP40000.2020.00041.

Prosvirnova, T., Batteux, M., Brameret, P.-A., Cherfi, A., Friedlhuber, T., Roussel, J.-M., Rauzy, A., 2013. The AltaRica 3.0 project for model-based safety assessment. IFAC Proc. Vol. 46 (22), 127–132. http://dx.doi.org/10.3182/20130904-3-UK-4041.00028, 4th IFAC Workshop on Dependable Control of Discrete Systems.

Rauber, J., Brendel, W., Bethge, M., 2017. Foolbox: A python toolbox to benchmark the robustness of machine learning models. arXiv preprint arXiv:1707.04131.

Rippel, O., Mertens, P., Merhof, D., 2021. Modeling the distribution of normal data in pre-trained deep features for anomaly detection. In: 2020 25th International Conference on Pattern Recognition. ICPR, IEEE, pp. 6726–6733.

SAE J3187_202202, 2022. System Theoretic Process Analysis (STPA) Recommended Practices for Evaluations of Automotive Related Safety-Critical Systems. SAE International, Warrendale, PA.

Sahner, R., Trivedi, K., Puliafito, A., 1995. Performance and Reliability Analysis of Computer Systems: An Example-Based Approach using the SHARPE Software Package. Springer US.

Salih, N.K., Satyanarayana, D., Alkalbani, A.S., Gopal, R., 2022. A survey on software/hardware fault injection tools and techniques. In: 2022 IEEE Symposium on Industrial Electronics & Applications. ISIEA, pp. 1–7. http://dx.doi.org/10.1109/ISIEA54517.2022.9873679.

Schierman, J.D., DeVore, M.D., Richards, N.D., Gandhi, N., Cooper, J., Horneman, K.R., Stoller, S.D., Smolka, S.A., 2015. Runtime Assurance Framework Development for Highly Adaptive Flight Control Systems. Barron Associates, Inc..

Schneider, C., Barker, A., Dobson, S.A., 2015. A survey of self-healing systems frameworks. Softw. - Pract. Exp. 45, 1375–1398.

SEBoK Editorial Board, 2022. In: Adcock (EIC), R. (Ed.), Guide to the Systems Engineering Body of Knowledge (SEBoK) v.2.7. The Trustees of the Stevens Institute of Technology, Hoboken, NJ.

Segismundo, A., Augusto Cauchick Miguel, P., 2008. Failure mode and effects analysis (FMEA) in the context of risk management in new product development: A case study in an automotive company. Int. J. Qual. Reliab. Manage. 25 (9), 899–912.

Shalev-Shwartz, S., Shammah, S., Shashua, A., 2017. On a formal model of safe and scalable self-driving cars. CoRR abs/1708.06374. arXiv:1708.06374.

Sharma, K.D., Srivastava, S., 2018. Failure mode and effect analysis (FMEA) implementation: a literature review. J. Adv. Res. Aeronaut. Space Sci. 5 (1–2), 1–17.

Shlyannikov, V., Yarullin, R., Ishtyryakov, I., 2020. Lifetime assessment for a cracked compressor disk based on the plastic stress intensity factor. Russ. Aeronaut. 63, 14–24.

Sifakis, J., 2018. System Design in the Era of IoT — Meeting the Autonomy Challenge. Electron. Proc. Theor. Comput. Sci. 272, 1–22. http://dx.doi.org/10.4204/EPTCS.272.1.

Sorin, D.J., 2009. Fault tolerant computer architecture. Synth. Lect. Comput. Archit. 4 (1), 1–104.

Spreafico, C., Russo, D., Rizzi, C., 2017. A state-of-the-art review of FMEA/FMECA including patents. Comput. Sci. Rev. 25, 19–28.

Srisakaokul, S., Wu, Z., Astorga, A., Alebiosu, O., Xie, T., 2018. Multiple-implementation testing of supervised learning software. In: AAAI Workshops. pp. 384–391.

Stapelberg, R.F., 2009. Handbook of Reliability, Availability, Maintainability and Safety in Engineering Design. Springer London.

Stewart, D., Liu, J.J., Cofer, D., Heimdahl, M., Whalen, M.W., Peterson, M., 2021. AADL-based safety analysis using formal methods applied to aircraft digital systems. Reliab. Eng. Syst. Saf. 213, 107649.

Sun, Y., Wu, M., Ruan, W., Huang, X., Kwiatkowska, M., Kroening, D., 2018. Concolic testing for deep neural networks. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. pp. 109–119.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R., 2013. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199.

Tazi, N., Châtelet, E., Bouzidi, Y., 2017. Using a hybrid cost-FMEA analysis for wind turbine reliability analysis. Energies 10 (3), 276.

Thomas, IV, J.P., 2013. Extending and Automating a Systems-Theoretic Hazard Analysis for Requirements Generation and Analysis (Ph.D. thesis). Massachusetts Institute of Technology.

Tian, J., Azarian, M.H., Pecht, M., 2014. Anomaly detection using self-organizing maps-based k-nearest neighbor algorithm. In: PHM Society European Conference. Vol. 2.

TimeNet homepage, 2018. https://timenet.tu-ilmenau.de/#/.

Trivedi, K., Bobbio, A., 2017. Reliability and Availability Engineering: Modeling, Analysis, and Applications. Cambridge University Press.

Tsai, T., Hari, S.K.S., Sullivan, M., Villa, O., Keckler, S.W., 2021. Nvbitfi: Dynamic fault injection for gpus. In: 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks. DSN, IEEE, pp. 284–291.

Wallace, M., 2005. Modular architectural representation and analysis of fault propagation and transformation. Electron. Notes Theor. Comput. Sci. 141 (3), 53–71.

WAYMO, 2018. Waymo reaches 5 million self-driven miles. https://blog.waymo.com/2019/08/waymo-reaches-5-million-self-driven.html.

Wegener, J., Baresel, A., Sthamer, H., 2001. Evolutionary test environment for automatic structural testing. Inf. Softw. Technol. 43 (14), 841–854. http://dx.doi.org/10.1016/S0950-5849(01)00190-2.

Weyns, D., 2017. Software engineering of self-adaptive systems: An organised tour and future challenges.

Weyns, D., 2021. An Introduction to Self-Adaptive Systems: A Contemporary Software Engineering Perspective. John Wiley & Sons, Ltd, http://dx.doi.org/10.1002/9781119574910.

Xie, M., 1991. Software Reliability Modelling, vol. 1, World Scientific.

Xing, L., Amari, S.V., 2015. Binary Decision Diagrams and Extensions for System Reliability Analysis. John Wiley & Sons.

Xing, J., Feng, C., Qian, X., Dai, P., 2012. A simple algorithm for sum of disjoint products. In: 2012 Proceedings Annual Reliability and Maintainability Symposium. pp. 1–5. http://dx.doi.org/10.1109/RAMS.2012.6175426.

Yoneyama, J., Artho, C., Tanabe, Y., Hagiya, M., 2019. Model-based network fault injection for IoT protocols. In: ENASE. pp. 201–209.

Yuan, X., Cai, B., Ma, Y., Zhang, J., Mulenga, K., Liu, Y., Chen, G., 2018. Reliability evaluation methodology of complex systems based on dynamic object-oriented Bayesian networks. IEEE Access 6, 11289–11300. http://dx.doi.org/10.1109/ACCESS.2018.2810386.

Zhang, J., Zhou, A., Sun, Q., Wang, S., Yang, F., 2018. Overview on fault tolerance strategies of composite service in service computing. Wirel. Commun. Mob. Comput. 2018.

Zheng, Y., Feng, Z., Hu, Z., Pei, K., 2021. MindFI: A fault injection tool for reliability assessment of MindSpore applicacions. In: 2021 IEEE International Symposium on Software Reliability Engineering Workshops. ISSREW, IEEE, pp. 235–238.

Zheng, Z., Zhou, T.C., Lyu, M.R., King, I., 2010. FTCloud: A component ranking framework for fault-tolerant cloud applications. In: Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on. IEEE, pp. 398–407.

Zhong, X., Ichchou, M., Saidi, A., 2010. Reliability assessment of complex mechatronic systems using a modified nonparametric belief propagation algorithm. Reliab. Eng. Syst. Saf. 95 (11), 1174–1185. http://dx.doi.org/10.1016/j.ress.2010.05.004.

Zhou, D., Tamir, Y., 2022. RRC: Responsive replicated containers. In: 2022 USENIX Annual Technical Conference. USENIX ATC 22.

Ziade, H., Ayoubi, R.A., Velazco, R., 2004. A survey on fault injection techniques. Int. Arab J. Inf. Technol. 1 (2), 171–186.

Zolghadri, A., 2017. The challenge of advanced model-based FDIR for real-world flight-critical applications. Eng. Appl. Artif. Intell. 68, http://dx.doi.org/10.1016/j.engappai.2017.10.012.