Computing (2021) 103:559–588
https://doi.org/10.1007/s00607-020-00874-x

SPECIAL ISSUE ARTICLE

# A conceptual framework for resilience: fundamental definitions, strategies and metrics

**Jesper Andersson[1] · Vincenzo Grassi[2] · Raffaela Mirandola[3] · Diego Perez-Palacin[1]**

Received: 14 February 2020 / Accepted: 17 November 2020 / Published online: 15 December 2020

**Abstract**
The *resilience* system property has become more and more relevant, mainly because of the increasing dependance on a rapidly growing number of software-intensive, complex, socio-technical systems, which are facing uncertainty about changes they are expected to experience during their life-cycle and ways to deal with them. Methodologies for the systematic design and validation of resilience for such systems are thus highly necessary, and require contributions from several different fields. This paper contributes to current resilience research by providing a conceptual framework intended to serve as a common ground for the development of such methodologies. Its main points are: the identification of the main categories of changes a system should face; a clear definition of the different facets of resilience one could want to achieve, expressed in terms of the system dynamics; a mapping of each of these facets to design strategies that are better suited to achieve it; and the corresponding identification of possible metrics that can be used to assess its achievement.

**Keywords** Resilience · Conceptual framework · Strategies and metrics

**Mathematics subject classification** 68U01 · 68N30 · 68U99

✉ Raffaela Mirandola
raffaela.mirandola@polimi.it

Jesper Andersson
jesper.andersson@lnu.se

Vincenzo Grassi
vincenzo.grassi@uniroma2.it

Diego Perez-Palacin
diego.perez@lnu.se

1 Linnaeus University, Växjö, Sweden

2 Universitá di Roma Tor Vergata, Rome, Italy

3 Politecnico di Milano, Milan, Italy

## 1 Introduction

In the last decade, *resilience* has become an increasingly relevant system property, because of the exponential growth (in number and dependence on them) of socio-technical systems that directly or indirectly may affect users' well-being. The unparalleled challenge for system engineers is to provide assurances for the behavior of such systems, in the face of uncertainties caused by the close interactions with their users and the environment, and *changes* they may need to adapt to, triggered by anticipated and unanticipated events in the system's environment, in the user needs and behaviors, and the system itself.

The concept of resilience was coined and developed in psychology to describe the human ability to cope with a crisis and to recover from it rapidly. Several other disciplines adopted the term over the years, including system safety [9], medicine [11], and human organization [3]. Widespread use in different disciplines has resulted in a situation where the term has several, sometimes incompatible or conflicting semantics. Woods [46] provides a comprehensive analysis of the different nuances of the resilience term.

If we put the magnifier glass on the ICT domain, we find a plethora of related terms that originate from different research communities such as the dependability, self-*, safety, and security communities: for example, resilience, robustness, adaptation, recovery, absorption, and flexibility, often without crisply defined relationships.

Consequently, it is not always clear which system aspect these different terms intend to capture, whether some are specializations (qualifications) of some other, or if some of them represent means for attaining a property indicated by another term. All in all, this causes difficulties for software engineers in the activities required to engineer resilient socio-technical systems and provide assurances for their behavior. With these challenges as starting point, the long-term research objective can be stated as:

> a methodology for systematic design, generation, and validation of resilient, software-intensive, socio-technical systems with assurances.

In this article, we report on our first results towards this goal. The main contribution is a *conceptual framework* intended to provide support for most development activities, which creates the right conditions for continued work towards the set research goal. Its main pillars are: the identification and characterization of the fundamental *change types* that affect system resilience; the principled definition of different facets of resilience, based on a *dynamic characterization* of resilience and corresponding to terms that concur with its definition; a mapping of each of these facets to design strategies that are better suited to achieve it; and the corresponding identification of possible metrics that can be used to assess its achievement. We use a simple case study to give concrete examples for the main elements characterizing our conceptual framework. Our contribution leverages the vast body of related work (e.g., [5,23,30,38,45,46]) that have already contributed to the general discussion on resilience. Many of these papers provide conceptual frameworks that assist in identifying the current state of the art, relationships among different approaches, and the promising research avenues.

We organize the paper as follows. In Sect. 2, we detail the objectives for a solution and identify the research gap. Section 3 introduces the proposed conceptual framework and the change types that impact system resilience. In Sects. 4 and 5 we present a discussion about existing resilience strategies, and a definition of a set of metrics related to the change types a system has to manage. In Sect. 6 we present the example case study, while we conclude and discuss future research in Sect. 7.

## 2 Problem, gap, and objectives for a solution

In this section, following the design science approach we use, we first identify and justify the problems that we observed. Continuing, we outline the solution we conjecture, and then describe objectives for this solution.

There are several significant challenges in software development linked to the development of recent years towards increasingly complex systems, where people interact with physical systems and become directly dependent on these systems behaving correctly and safely. Another trend that adds further complexity is that systems are, to a greater extent open, that is, they change dynamically during the life cycle.

The biggest challenge for software engineers today is to guarantee that these complex systems are dependable under the invariant that the systems and their environment change dynamically. Littlewood and Strigini [25] informally define *dependability* as a set of system properties "that allows us to rely on a system functioning as required." Laprie extends this in his definition of *resilience* as "dependability when facing changes." [23].

Software engineers who are supposed to provide assurances for a system's resilience must take into account all properties that can affect a system's ability to function as expected, which means that experts on different properties will be active. The system design will use experiences and solutions from several different specialty areas, such as accessibility, security, performance, and reliability, which must be co-designed, implemented, and verified to provide assurances for correct behavior.

A solution to the problems and challenges outlined above is a methodology for system resilience. A methodology that:

- leverages specialty areas that contribute to resilience.
- provide support for the complete system life-cycle.
- provide extensive support for architectural reasoning and assurances for the resilience property.
- support seamless offline and online adaptation and evolution [2].

A first step and objective for this solution is to define a common ground for the specialty areas. This common ground will enable some key practices in software engineering that will speed up processes, enable reuse [22] and improve process and product quality. The objectives for a common ground include:

- a conceptual framework based on the principled definitions of terms that concur with the resilience definition;
- a characterization of the change types affecting the system resilience;
- a dynamic characterization of resilience in terms of the types of change the system has to cope with;
- a discussion about strategies to achieve resilience;

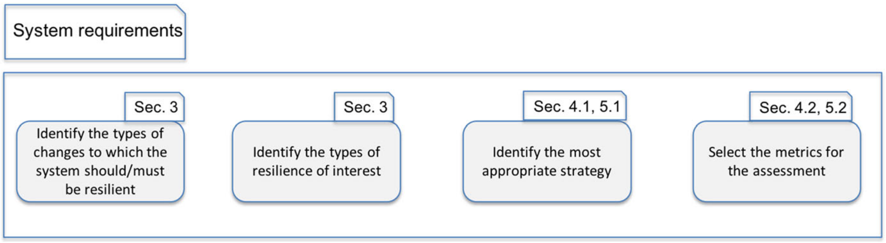

System requirements
Sec. 3
Identify the types of changes to which the system should/must be resilient
Sec. 3
Identify the types of resilience of interest
Sec. 4.1, 5.1
Identify the most appropriate strategy
Sec. 4.2, 5.2
Select the metrics for the assessment

**Fig. 1** Conceptual framework main pillars

– a definition of a set of metrics that can be used to assess the system resilience according to the different changes the system has to handle, and the goals it intends to achieve.

To pursue our goal, we contribute in this direction, by distilling and presenting concepts from the current body of work in a unified and concise way. In particular, for the definition of our conceptual framework, we use the ideas expressed in [5,23, 38] as our starting point. They are the result of discussions mainly belonging to the dependability and self-organizing systems communities. In particular, we leverage Laprie's definition [23] that defines resilience as an extension of dependability when facing changes. Besides, we also refer to the general discussion reported in [46]. Outside the ICT domain, other works that have presented clarifying models, conceptual frameworks and possible metrics for resilience assessment (and that have inspired some of our contributions) are, for example, [12,19,29,35]. We refer to (and briefly comment) still other works in the next sections of this paper.

Hereafter, we concisely illustrate the main pillars of the proposed conceptual framework as a roadmap (depicted in Fig. 1) to be applied to understand and evaluate the system resilience.

We assume that a conventional process of requirements discovery and elicitation has been applied to obtain a set of system requirements including resilience aspects. The first step to be undertaken to apply our proposed framework is to identify the types of changes to which the system must/should be resilient. We propose in Sect. 3 a classification of possible types of changes that lead to a modification of the system or to a change in the system acceptance criteria. Once the types of changes have been identified, it is necessary to understand the corresponding kind of resilience that is expected from the system. A detailed characterization of resilience declined according to the possible types of changes is presented in Sect. 3. For each type of resilience is then key understanding possible strategies for enabling that type of resilience in a system (Sects. 4.1 and 5.1) and metrics and measurements strategies that can be used for the resilience assessment (Sects. 4.2 and 5.2).

## 3 A conceptual framework for characterizing resilience in ICT systems

In this section, we introduce and briefly explain terms and concepts that capture essential aspects of the ICT systems resilience discourse. Using a broad definition of *resilience* as a starting point, we characterize resilience using these terms and concepts and describe the basic properties of the change types that affect system resilience.

## 3.1 Basic terms and concepts

**Resilience** In the following we conform to the Laprie's definition [23].

**Definition** Resilience is defined as the persistence of dependability when facing changes.

This definition refers to the *dependability* concept, which is a fundament in a conceptual framework elaborated over several years within the dependable computing community [5]. It defines dependability as: "The ability to deliver service that can justifiably be trusted." or, alternatively:"The dependability of a system is the ability to avoid service failures that are more frequent and more severe than acceptable."

From these definitions, it is clear that resilience is a broader concept than dependability due to an increase in the number of event types that may affect the system property. Dependability concerns a system's ability to deliver satisfactory service in the presence of "negative" events, such as, faults and even failures. Resilience is more general as it is concerned with a system's ability to deliver satisfactory service in the presence of *changes*. Changes are not necessarily negative events, for example, in ubiquitous systems where a continuous change in the number and type of interacting entities is a rule rather than an exception.

**System and Environment** By *System* we mean a broad notion encompassing hardware and software systems, humans, and the physical world with its natural phenomena in which the software and hardware systems are situated. In the research reported herein, we focus on ICT systems consisting of hardware and software components.
The systems we consider are *structured systems* that consists of a collection of interacting components, where each component by itself constitutes a system. This definition is recursively applicable until we reach a decomposition level where further decomposition is not relevant for the given context. Besides interacting with other components that are part of the same system, a system also interacts with systems in the system's *environment*. The observers perspective and context define the system-environment boundary. The system interacts and affects the environment, and it is in the environment that observers may evaluate the system effects on it.

**System state** The *system state* is the collection of attributes required for describing a system and its behavior.
Hence, a specific state can be modeled as a vector $\boldsymbol{\sigma}$ belonging to some $n$-dimensional state space $\boldsymbol{\Sigma}$. This simplified state notion encompasses parameters and attributes characterizing both a system and its environment.

**State classification** An *acceptance criterion* $\theta$ is a set of constraints and relationships defined on the system state that allows the identification of the subset of the system state space $\boldsymbol{\Sigma}$ consisting of all those states where the service delivered by the system

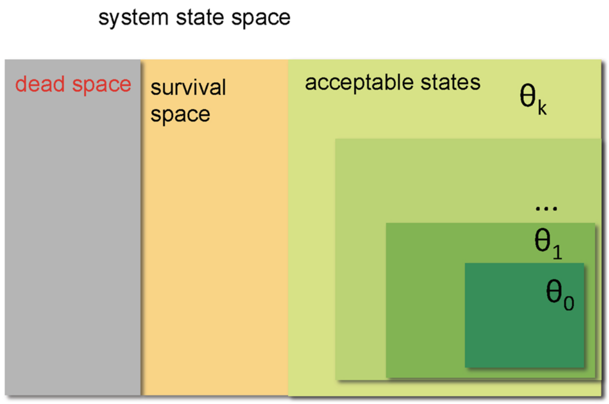


**Fig. 2** States classification (adapted from [38])

can be considered correct and acceptable according to $\theta$. We call this subset the set of *acceptable states* with respect to $\theta$, and denote it by $\theta(\boldsymbol{\Sigma})$.
In general, a number of acceptance criteria $\theta_0, \theta_1, \ldots, \theta_k$ could be defined for a given system, such that $\theta_0(\boldsymbol{\Sigma}) \subseteq \theta_1(\boldsymbol{\Sigma}) \subseteq \cdots \theta_k(\boldsymbol{\Sigma}) \subseteq \boldsymbol{\Sigma}$. The case $k \geq 1$ thus allows considering a series of progressively less stringent acceptance criteria, which can be used in situations where we want to distinguish different levels of more or less degraded but still acceptable performance. Otherwise, the case $k = 0$ represents an on-off situation, where the system state is either acceptable or not acceptable. For a comparison, the discussion in [38] assumes $k = 1$, where $\theta_0(\boldsymbol{\Sigma})$ and $\theta_1(\boldsymbol{\Sigma})$ are called *target space* and *acceptable space*, respectively. On the other hand, the discussion in [5] basically assumes $k = 0$, with $\boldsymbol{\Sigma} \backslash \theta_0(\boldsymbol{\Sigma})$ the set of *error states*.

To fully characterize the system behavior, we introduce two additional subsets of $\boldsymbol{\Sigma}$, denoted by $\theta_s(\boldsymbol{\Sigma})$ and $\theta_d(\boldsymbol{\Sigma})$, such that $\boldsymbol{\Sigma} = \theta_k(\boldsymbol{\Sigma}) \bigcup \theta_s(\boldsymbol{\Sigma}) \bigcup \theta_d(\boldsymbol{\Sigma})$, and $\theta_x(\boldsymbol{\Sigma}) \bigcap \theta_y(\boldsymbol{\Sigma}) = \emptyset$, for any $x, y \in \{k, s, d\}, x \neq y$. Following [38], we call them the *survival space* and *dead space*, respectively.

The survival space $\theta_s(\boldsymbol{\Sigma})$ includes all those states where the service delivered by the system is not acceptable, but for which a sequence of internally or externally initiated corrective actions exists, which bring the system back to a state $\boldsymbol{\sigma} \in \theta_i(\boldsymbol{\Sigma}), 0 \leq i \leq k$. The dead space $\theta_d(\boldsymbol{\Sigma})$ includes all states where the delivered service is not acceptable and that preclude the possibility of returning to an acceptable state. Figure 2 depicts the state classification.

## 3.2 A dynamic characterization of resilience

The resilience definition given in the previous subsection (analogously to the dependability definition from which it is derived) is intended to represent a general and global concept that subsumes several more specific concepts concerning one or more of its facets. In this section, we answer the question: what do we expect from a "resilient system"? Any answer to this question reflects which incarnation of the different resilience concepts it originates from. Further, it will require the adoption of different design and implementation strategies to achieve resilience and the application of different metrics for its measurement.

To this end, we revisit the general definition of resilience using the definitions from the related domains as a prism. Further, we suggest an experimental characterization of the resilience incarnations in terms of system dynamics defined by state transitions and state trajectories.

For a start, the considered resilience definition stresses that it is a property strongly related with the trust we can have in the system ability to remain inside the boundary of some set $\theta_i(\boldsymbol{\Sigma}), 0 \leq i \leq k$, despite the occurrence of events, generically called "changes", that may challenge this ability. Changes are called "disturbances" in [38], and "faults" in [5].

We can distinguish two main kinds of change events that may force a system to cross the boundary of an acceptable states set:

– *operational changes*: changes that lead to a modification of the system and/or environment state, denoted as a function $\delta : \boldsymbol{\Sigma} \rightarrow \boldsymbol{\Sigma}$. Examples of this kind of events could be a change in the load and/or profile of service requests addressed to a system, a fault of some internal component of the system, the appearance/disappearance of resources in the system environment. Such changes lead to a border crossing if, given a state $\boldsymbol{\sigma} \in \theta_i(\boldsymbol{\Sigma})$, we have $\delta(\boldsymbol{\sigma}) \notin \theta_i(\boldsymbol{\Sigma})$.
– *evolutionary changes*: changes that lead to a modification of the acceptance criterion, denoted as a function $\rho : \boldsymbol{\Theta} \rightarrow \boldsymbol{\Theta}$, where $\boldsymbol{\Theta}$ generically denotes the set of possible acceptance criteria. Examples of this kind of events could be a change in the user preferences or requirements, which causes the addition of new criteria, and/or the removal or modification of old criteria. Such changes lead to a border crossing if, given a state $\boldsymbol{\sigma} \in \theta_i(\boldsymbol{\Sigma})$, we have $\boldsymbol{\sigma} \notin \rho(\theta_i)(\boldsymbol{\Sigma})$.

We may use these change types to identify several resilience variants. We first consider resilience with respect to a given set **OC** of operational changes, which could affect a system or its environment. Then, we consider resilience with respect to a given set **EC** of evolutionary changes.

The proposed resilience classification, with respect to **OC** and a given set of acceptance criteria $\theta_0, \theta_1, \ldots, \theta_k$, depends on which kind of border crossing these changes are able to induce. Besides ideas expressed in [5,38], this classification is also inspired by the discussion in [46].

**Definition** A system is *robust* with respect to **OC** and an acceptance criterion $\theta_i$, if for any $\delta \in \mathbf{OC}$ and $\boldsymbol{\sigma} \in \theta_i(\boldsymbol{\Sigma})$, it is $\delta(\boldsymbol{\sigma}) \in \theta_i(\boldsymbol{\Sigma})$.

This means that a robust system with respect to **OC** never crosses the boundary of the set of acceptable states $\theta_i(\boldsymbol{\Sigma})$. This property is called "strong robustness" in [38], and "robustness" (alias "resilience(2)") in [46]. An illustration of this type of resilience is given in Fig. 3a.

**Definition** A system is *gracefully* degradable with respect to **OC** and an acceptance criterion $\theta_i$, with $i < k$, if for any $\delta \in \mathbf{OC}$ and $\boldsymbol{\sigma} \in \theta_i(\boldsymbol{\Sigma})$, it is $\delta(\boldsymbol{\sigma}) \in \theta_k(\boldsymbol{\Sigma})$.

Graceful degradability is thus a weaker property with respect to robustness, however, it retains the idea that the system will always be able to deliver some kind of

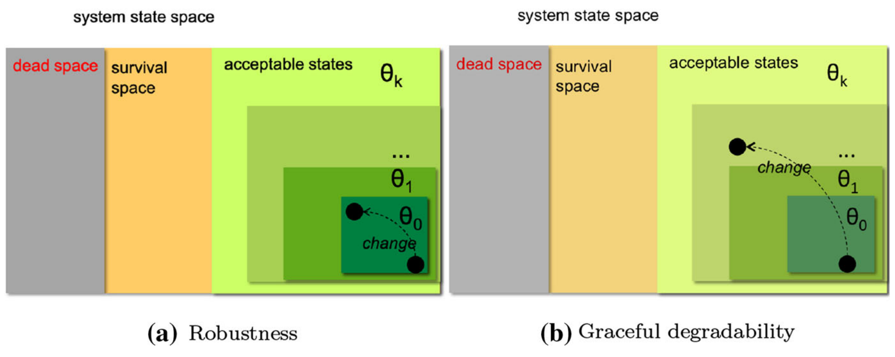


**Fig. 3** Resilience types with acceptable states

**Table 1** Resilience with respect to OC

| | | Entering non-acceptable states | |
|---|---|---|---|
| | | No | Yes |
| Degradation | No | *robustness* | *recoverability to best* |
| | Yes | *graceful degradability* | *recoverability* |

minimally acceptable service and never enter a non acceptable state. This property is called "weak robustness" in [38], but limited to the case $i = 0$ and $k = 1$. It also partially resembles the "graceful extensibility" (alias "resilience(3)") in [46]. Figure 3b depicts the states in case of graceful degradability.

**Definition** A system is *recoverable* with respect to **OC** and an acceptance criterion $\theta_i$, if for any $\delta \in \mathbf{OC}$ and $\boldsymbol{\sigma} \in \theta_i(\boldsymbol{\Sigma})$, it is $\delta(\boldsymbol{\sigma}) \in \theta_k(\boldsymbol{\Sigma}) \cup \theta_s(\boldsymbol{\Sigma})$.

Recoverability thus implies that the system could temporarily enter states where the delivered service is not acceptable, but has access to sufficient capabilities that enable it to return to an acceptable state by itself or by external control. This property is called "adaptivity/adaptability" in [38]. It is also related to the "rebound" (alias "resilience(1)") property in [46]. This type of resilience is illustrated in Fig. 4a.

Table 1 summarizes the types of resilience to **OC** discussed above in terms of tolerance to the occurrence of non-acceptable or degraded states. Besides the three main types of resilience we have identified, the table evidences an additional special case, where temporarily entering non-acceptable states is considered acceptable provided that the system is able to recover to the optimal states $\theta_0(\boldsymbol{\Sigma})$.

Let us now consider a given set **EC** of evolutionary changes. We can distinguish some different scenarios:

**Definition** **EC** is a *relaxation* of $\theta_k$, when for any $\rho \in \mathbf{EC}$ it is:
$\theta_k(\boldsymbol{\Sigma}) \bigcap \rho(\theta_k)(\boldsymbol{\Sigma}) = \theta_k(\boldsymbol{\Sigma})$.

In this case, a system that is robust/gracefully degradable/recoverable with respect to a given set of operational changes **OC** retains the same kind of resilience in the new scenario generated by the introduction of **EC**.

**Definition** **EC** is a *restriction* of $\theta_k$, when for any $\rho \in \mathbf{EC}$ it is:
$\theta_k(\boldsymbol{\Sigma}) \bigcap \rho(\theta_k)(\boldsymbol{\Sigma}) = \rho(\theta_k)(\boldsymbol{\Sigma})$.

In this case, a system that is robust for a given set of operational changes **OC** loses this resilience property. It cannot guarantee that it can remain within the boundary of the narrower set of acceptable states. On the other hand, a system that is gracefully degradable/recoverable for **OC** retains the same kind of resilience also for the new acceptance criteria defined by **EC**, as it has the built-in capability of maintaining or returning to states where at least a degraded version of the acceptance criteria defined by **EC** holds.

**Definition** **EC** is a *variation* of $\theta_k$ when for any $\rho \in \mathbf{EC}$ it is:
$\theta_k(\boldsymbol{\Sigma}) \bigcap \rho(\theta_k)(\boldsymbol{\Sigma}) \neq \rho(\theta_k)(\boldsymbol{\Sigma})$ and $\theta_k(\boldsymbol{\Sigma}) \bigcap \rho(\theta_k)(\boldsymbol{\Sigma}) \neq \theta_k(\boldsymbol{\Sigma})$. Therefore, a variation introduces a partially or totally new set of acceptance criteria.

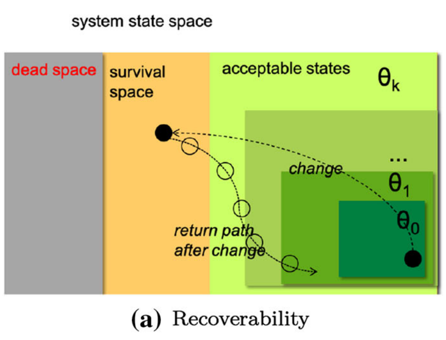
This implies that at least some of the new acceptable states are outside the borders of the old set of acceptable states. In the extreme case, all the new states are outside the borders of the old states, when $\theta_k(\boldsymbol{\Sigma}) \bigcap \rho(\theta_k)(\boldsymbol{\Sigma}) = \emptyset$. As a consequence, in this scenario it does not make sense to try to achieve either robustness or graceful degradation: it is an intrinsic property of this scenario that a given system state that was acceptable before the change caused by **EC** is no longer acceptable (not even as a "degraded" state). The system will thus necessarily experience a permanence in a non-acceptable state for some time. If the system after some time in this condition can change its operations and thereby reach and stay within the new set of acceptable states, the system is resilient to these changes. This behavior resembles *recoverability* discussed above. However, it requires a different kind of capability compared to recoverability. Recoverability realizes the idea that a system always can bounce back to a visited condition, while the scenario we are considering requires a system that is capable of reaching a previously unvisited condition. The following definition intends to capture this different perspective on resilience.

**Definition** A system is *flexible* when it is resilient to EC variations.

This property is similarly called "flexibility" in [38]. It is also related with the "graceful extensibility" (alias "resilience(3)") and "sustained adaptability" (alias "resilience(4)") properties in [46]. Figure 4b illustrates the system state space in the case of flexible systems.

We summarize in Table 2 the different perspectives on resilience we have included in our classification, together with their counterparts in [38,46].

To conclude this characterization of the resilience concept, we note that our discussion seems to define a hierarchy, with robustness at the top and recoverability and flexibility at the bottom. We want to point out that this hierarchy is only apparent, as it actually holds only under the assumption of an invariant set of changes for all the given definitions; the relative merit of each kind of resilience depends instead on several factors that include, for example, a trade-off among the cost to stay in degraded or non-acceptable states, the cost to provide a system that may never enter these states, and the variety of changes the system is able to cope with. This kind of considerations,



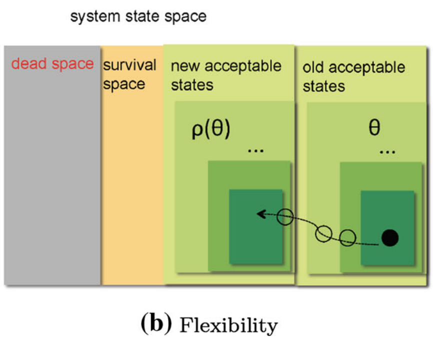


**Fig. 4** Resilience types that reach non-acceptable states

**Table 2** Resilience classification

| Our proposal | Schmeck et al. [38] | Woods [46] |
|---|---|---|
| Robustness | Strong robustness | Resilience(2)/robustness |
| Graceful degradability | Weak robustness | Resilience(3)/graceful extensibility |
| Recoverability | Adaptivity/adaptability | Resilience(1) |
| Flexibility | Flexibility | Resilience(3)/graceful extensibility and resilience(4)/sustained adaptability |

where "cost" could encompass several aspects including economic and human, could lead designers to consider as more viable and effective an apparently weaker kind of resilience. Moreover, as pointed out in [46], we should also consider that the over-provisioning implied by robustness for some set of changes, may lead to increased vulnerability to other changes not included in the set under consideration.

### 3.3 Basic properties of change that affect the system resilience

In the previous subsection, we have characterized resilience in terms of inter or intra state-set transitions, triggered by generic "changes" that affect a system. We want to make the characterization more precise to facilitate the assessment of resilience. To this end, we can identify two dimensions, if we study change events from the system perspective: readiness and persistence. These two dimensions are orthogonal to the ones discussed in the previous section.

– the *readiness* of the system to handle a given change; in this case we distinguish between *expected* and *unexpected* changes, and, within the expected, we further distinguish between *handled* and *unhandled* changes;
– the *persistence* of the impact of a given change on the system; in this case we distinguish between *transient* and *permanent* changes.

Figure 5 depicts this classification of changes. We further discuss how this characterization of changes fits into the resilience characterization introduced above.

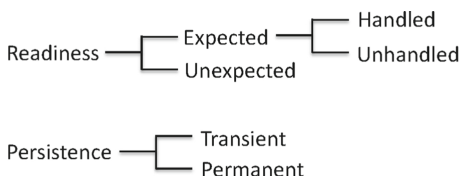


**Fig. 5** Changes according to the their persistence and readiness to be handled

**Expected versus unexpected changes** Expected and handled changes are changes that are part of a system's "normal" operation, in the sense that the system includes hardware and software resources that enable it to manage the changes. These changes include those belonging to the system's nominal behavior, for example, changes in the value read within the working range of a sensor, and "undesired" changes, for example failures. Depending on which design decisions designers make, the system handles these changes differently. Aligned to the classification presented herein, designers make the system robust, gracefully degradable, or recoverable concerning the changes.

Expected unhandled changes are those changes that are foreseeable and identified, but for which no system provisioning is in place to manage them. The consequence is that if such an event occurs, it is likely it brings the system into an unacceptable state, either in the survival $\theta_s(\boldsymbol{\Sigma})$ or the dead $\theta_d(\boldsymbol{\Sigma})$ space. Designers may decide not to handle some types of changes during the system development. The rationale for not handling a foreseeable change can be a relatively low occurrence frequency or complicated and costly techniques to introduce system mechanisms that handle that change type. The consequence of these decisions is that the system will lack mechanisms that retain or return the system to an acceptable state automatically. There may however exist protocols that system operators may follow to return the system to an acceptable state. If there is a protocol for recovering from an identified change, then the change moves the system to a state in the survival space. If there is no such protocol, the change moves the system to a state in the dead space.

If designers do not identify a possible change, it results in possible unexpected changes at runtime, the so-called *surprises* [46,47].[1] An unexpected change may move the system to any subspace in the state space, acceptable, survival, or dead spaces. The resilience classification discussed above is not equipped to manage this type of change as it would require a system that can reason about the effect of situations that it is unaware of and possibly identify a protocol for returning the system to an acceptable state. In some cases, the system may already be resilient (robust, gracefully degradable or recoverable, as defined in Sect. 3.2). Apart from these cases, the additional design and development efforts that are required to make the system able to cope with the new type of changes can be reduced if the system has been designed and implemented with a good degree of *flexibility*, as defined in Sect. 3.2, and as remarked also in [46] ("graceful extensibility" (alias "resilience(3)") and "sustained adaptability" (alias "resilience(4)") properties).

**Permanent versus transient changes** Following the distinction of fault persistence in [5], we can distinguish two types of change. *Permanent* changes are changes that

[1] A possible contribution to the formalization of this concept can be found in [8].

do not disappear unless some corrective action takes place. *Transient* changes are changes where the system eventually returns to an acceptable state without taking any action. An example of transient change is a power outage that affects a network router. When the power comes back, the router returns to function. Another example is when a software component fails to establish a connection to a database due to multiple concurrent requests. When the load decreases, the component may connect to the database. An example of permanent change is the addition of a new system requirement to be satisfied that is not covered by the existing ones.

When a designer identifies a transient change, the decision of whether handling it or not is a tradeoff between the cost and occurrence of the effect, and the cost of handling the change. If the change is unhandled, the system is intrinsically recoverable (as defined in Sect. 3.2) for the change, with a recovery period duration that corresponds to the duration of the change, until it disappears. A system should instead always handle permanent changes; that is, it should to be resilient for such changes.

## 4 Design strategies and resilience metrics for operational changes

We recall from Sect. 3 that by *operational changes* we mean those changes that modify the system internal or environment state, thus possibly impairing the system ability to fulfill a given set of acceptance criteria, which instead remain unchanged. Making a system resilient to this type of changes introduces many challenges from a design and implementation perspective. In this section, we discuss strategies for enabling resilience in a system, and resilience metrics and measurement strategies that can be used in resilience assessment for operational changes.

### 4.1 Resilience strategies

There is a general consensus across different research fields (e.g., [12,19,35]) that strategies aimed at making a system resilient to operational changes can be understood in terms of the following three goals, which can be pursued independently or in combination:

*reduced failure probability*: this goal concerns the mitigation of hazards, by preventing the possibility that the occurrence of a change leads to a system failure;
*reduced consequences from failures*: this goal concerns the containment of the severity of the negative consequences experienced by a system when a failure occurs because of some change;
*reduced time to recovery*: this goal concerns the speed at which a system can recover from a failure, restoring its performance to some "normal" level.

These goals can be put quite naturally in correspondence with the first three resilience variants we have identified in Sect. 3, mainly concerning operational changes: *robustness*, *graceful degradability* and *recoverability*. In the following we briefly discuss strategies to achieve those three kinds of resilience, highlighting their relationships with the three goals outlined above.

**Resilience as robustness** This property is strictly related with the *reduced failure probability* goal. We may achieve it by utilizing *redundancy* techniques. These often do not require explicit detection mechanisms for the occurrence of changes or mechanisms that precisely identify the change type (e.g., fault masking using parallel active redundancy with majority voting).
An alternative strategy is *intrinsic algorithmic and structural* system properties that can manage the change within the system's "normal flow" of events. One example of such systems is the self-organizing systems [17] that do not require an explicit mechanism that detects the occurrence of a change.
A third strategy is *proactive (self-)adaptation* that uses forecasting to anticipate possible changes before their occurrence and enacts corrective actions that prevent undesired state changes; as a consequence, these latter techniques do require the identification of the type of change that will occur.

**Resilience as graceful degradability** This property is related with the *reduced consequences from failures* goal. Indeed, this goal can be achieved by trying to keep to a minimum the "distance" between the "ideal" acceptance criterion $\theta_0$ and the "worst" acceptance criterion $\theta_i$, $i > 0$ fulfilled by the system because of its performance degradation after a change occurrence. Also for this kind of resilience, like for robustness, designers may utilize *redundancy* techniques to achieve it. These techniques are generally different from the techniques discussed above for robustness, but they share the same advantages of not generally requiring explicit detection and identification of a change occurrence (e.g., a servers cluster that continues to work at reduced capacity when some server fails, irrespective of the actual cause of server failure).
An alternative strategy is *reactive (self-)adaptation* that in this instance, identifies the change that has occurred and adapts the service or service quality. For example, a video streaming service detects a change in the available bandwidth and reduces the frame rate to be able to continue the service delivery.

**Resilience as recoverability** This property is related with the *reduced time to recovery* goal. Indeed, this goal can be achieved by minimizing the time possibly spent in states belonging to $\theta_s(\boldsymbol{\Sigma})$, or some $\theta_i(\boldsymbol{\Sigma})$, $i > 0$, before restoring the system to states in $\theta_0(\boldsymbol{\Sigma})$. The recoverability property is generally achieved utilizing *reactive (self-)adaptation* techniques, which span commoditized techniques like checkpoint-rollback-recovery in database systems and alternative strategies based on, for example, machine-learning methodologies to identify a suitable adaptation plan that restores the system to a correct operational status.

## 4.2 Resilience metrics for operational changes

The critical role resilience plays in ICT systems elevates the importance of metrics and indicators that provide a quantitative evaluation of resilience. These metrics assist designers and other decision-making stakeholders in obtaining an understanding of a system's resilience status. Hence, they are better prepared to identify, plan, and prioritize activities that improve system resilience. In the past, several efforts have addressed this area and proposed several approaches.

In the following, we present possible metrics that can be used for system assessment with respect to the three perspectives on resilience discussed above, and we detail how their values can be computed when using the reasoning framework in Sect. 3. In particular, referring to the resilience characterization given in Sect. 3.2, these metrics are observation-based; that is, they monitor the system's operational state trajectories. The trajectories may visit different parts of the system's state space. With these mechanisms in place, we may collect information and express system resilience in terms of whether or not it has visited some parts of the state space, and if so the duration of the visit and which states it visited in $\theta_i(\boldsymbol{\Sigma})$, for some $i \in \{0, 1, \ldots, k\}$, or in $\theta_s(\boldsymbol{\Sigma})$ or $\theta_d(\boldsymbol{\Sigma})$. We point out that all the metrics introduced in the following subsections can be considered as "descriptors" of the observed state trajectories. If the system dynamics is modeled by means of some stochastic model, then the proposed metrics are actually random variables, whose moments or probability distribution can be used as actual resilience metrics.

### 4.2.1 Metrics for quantifying the ability to prevent failure

According to the discussion in Sect. 4.1, these metrics are related with the *reduced failure probability* goal, and the *robustness* view of resilience, and are intended to measure the system ability to prevent disruptive consequences when some change occurs. Broadly speaking, this property concerns the *continuity* of the system correct service. Referring to the resilience literature, this property is referred to as "reduced failure probabilities" [12], also called "mitigation" [35].

Within our proposed reasoning framework, we give below examples of two possible metrics of this type, where the former is time-dependent, while the latter can be considered as a time-independent metric.

**A time-dependent "failure prevention" measure** Let $f_1 : \boldsymbol{\Sigma} \rightarrow \Re$ be a function that maps the system state space to the set of real numbers defined as:

$$f_1(\boldsymbol{\sigma}) = \begin{cases} 0 & \text{if } \boldsymbol{\sigma} \in \theta_k(\boldsymbol{\Sigma}) \\ 1 & \text{if } \boldsymbol{\sigma} \in (\boldsymbol{\Sigma} \backslash \theta_k(\boldsymbol{\Sigma})) \end{cases} \tag{1}$$

A "failure prevention" metric can be defined as:

$$\phi(T_0, T_1) = \int_{T_0}^{T_1} f_1(\boldsymbol{\sigma}(t))dt \tag{2}$$

where the time instants $T_0$ and $T_1$ denote, respectively, the start and stop points for system observation (it could be $T_0 = 0$ and/or $T_1 = \infty$), and $\boldsymbol{\sigma}(t)$ denotes the system state at time $t \in (T_0, T_1)$. $\phi(T_0, T_1)$ thus measures the time spent in the interval $(T_0, T_1)$ in non-acceptable states (i.e. corresponding to a system failure). In particular, $\phi(T_0, T_1) = 0$ indicates that the system has not experienced any unacceptable degradation (failure) in the observation interval $(T_0, T_1)$.

In the dependability domain, metrics of this kind are those measuring the system *reliability* (e.g., the mean time to failure (MTTF), or the probability of no failure in some time interval $[0, T]$, where $T$ is the length of the system mission time).

**A time-independent "failure prevention" measure** Let $f_2 : \boldsymbol{\Sigma^2} \rightarrow \Re$ be a function mapping pairs of system states to the set of real numbers defined as:

$$f_2(\boldsymbol{\sigma_1}, \boldsymbol{\sigma_2}) = \begin{cases} 0 & \text{if } \arg\min_i(\boldsymbol{\sigma_1} \in \theta_i(\boldsymbol{\Sigma})) \geq \arg\min_j(\boldsymbol{\sigma_2} \in \theta_j(\boldsymbol{\Sigma})) \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

that is, $f_2(\boldsymbol{\sigma_1}, \boldsymbol{\sigma_2})$ returns 1 iff $\boldsymbol{\sigma_1}$ satisfies more stringent acceptance criteria than $\boldsymbol{\sigma_2}$, otherwise it returns 0.

Let us focus on a specific $\delta_e \in \mathbf{OC}$, $\delta_e : \boldsymbol{\Sigma} \rightarrow \boldsymbol{\Sigma}$. According to the definitions in Sect. 3, $\delta_e$ represents how some disruptive event $e$ changes the state of the system depending on the current state when the event occurs.

A "failure prevention" metric with respect to event $e$ can be defined as:

$$\gamma(\boldsymbol{\sigma}, e) = f_2(\boldsymbol{\sigma}, \delta_e(\boldsymbol{\sigma})) \tag{4}$$

Indeed, $f(\boldsymbol{\sigma}, \delta_e(\boldsymbol{\sigma}))$ returns 1 if event $e$ has $deteriorated$ the system, i.e, it has made the system able to satisfy only *less restrictive* acceptance criteria, and is thus an indicator of the possible negative impact of event $e$ when the system is in some state $\boldsymbol{\sigma}$.

As an example, in a probabilistic setting, if we denote by $p(\boldsymbol{\sigma})$ the probability for the system of being in state $\boldsymbol{\sigma}$ (e.g., it could be calculated as the steady state probability of state $\boldsymbol{\sigma}$, or the probability of being in state $\boldsymbol{\sigma}$ during a specific time interval), then a possible probabilistic *failure prevention* metric could be derived from $\gamma(\boldsymbol{\sigma}, e)$:

$$1 - \sum_{\forall \sigma \in \boldsymbol{\Sigma}} \gamma(\boldsymbol{\sigma}, e) p(\boldsymbol{\sigma}) \tag{5}$$

Indeed, it is the probability that the event $e$ does not degrade the system performance, in whatever system state it occurs. The greater its value, the greater it can be considered the system resilience to this event.

### 4.2.2 Metrics for quantifying the consequences from failure

According to the discussion in Sect. 4.1, these metrics are related with the *reduced consequences from failures* goal, and the *graceful degradation* view of resilience, and are intended to measure the system ability to contain or reduce the negative consequences caused by the occurrence of some change. Broadly speaking, this property concerns the *overall accumulated "quality"* (or *"degradation"*) of the service delivered by the system. Referring to the resilience literature, this property is referred to as *reduced consequences from failures* in [12], *"static" resilience* in [35], *level of recovery* in [19], *absorption* and *adaptation* in [29].

Within our proposed reasoning framework, we give below examples of possible metrics for this property. Referring to the dependability domain, metrics of this kind are those measuring the system *performability* (e.g., average quality accumulated in a time interval, assuming that different quality levels are associated with states in different sets $\theta_i(\boldsymbol{\Sigma})$). To this end, we introduce the following "reward" function $r : \boldsymbol{\Sigma} \rightarrow \Re$ defined as:

$$r(\boldsymbol{\sigma}) = d_i \qquad \text{if } \boldsymbol{\sigma} \in \theta_i(\boldsymbol{\Sigma}) \tag{6}$$

with $0 = d_0 \leq d_1 \leq \cdots \leq d_k \leq d_s \leq d_d$, where each $d_i$ is a measure of the amount of suffered degradation when the system is in a state $\boldsymbol{\sigma} \in \theta_i(\boldsymbol{\Sigma})$.

**Cumulative amount of degradation** Let us define $T_{start}$ as the time when a disruptive event occurs, and $T_{end}$ as the time when the system is restored to a fully functional state.

Let $\kappa(T_{start}, T_{end})$ be defined as:

$$\kappa(T_{start}, T_{end}) = \int_{T_{start}}^{T_{end}} r(\boldsymbol{\sigma}(t))dt \tag{7}$$

$\kappa(T_{start}, T_{end})$ is thus the *cumulative amount of degradation* in $(T_{start}, T_{end})$. The smaller its value, the smaller the overall degradation suffered by the system.

However, $\kappa(T_{start}, T_{end})$ does not allow discriminating between a system that, after a disruptive event occurrence, experiences a very large disruption but then quickly recovers to a "quasi-normal" state, and a system that instead experiences a milder disruption but then recovers much more slowly. Depending on the considered domain, these two different behaviors could be more or less preferable. The following metrics are intended to help in discriminating between them.

**Degradation severity** Differently for the previous metric $\kappa(T_{start}, T_{end})$, which measures the cumulative (negative) impact of a change, the following metrics focus instead on the peak impact.

A first metric of this kind could be the *maximum amount of degradation* in $(T_{start}, T_{end})$, defined as:

$$\xi(T_{start}, T_{end}) = \max_{t \in (T_{start}, T_{end})} \{r(\boldsymbol{\sigma}(t))\} \tag{8}$$

Another possible metric of the same kind could be defined as follows. Let $g : \boldsymbol{\Sigma} \times \Re \rightarrow \Re$ be defined as.

$$g(\boldsymbol{\sigma}, d) = \begin{cases} 0 & \text{if } r(\boldsymbol{\sigma}) < d \\ 1 & otherwise \end{cases} \tag{9}$$

where $r()$ is defined as in Eq. (6).

Then, a disruption severity metric can be defined as:

$$\zeta(T_{start}, T_{end}) = \int_{T_{start}}^{T_{end}} g(\boldsymbol{\sigma}(t))dt \tag{10}$$

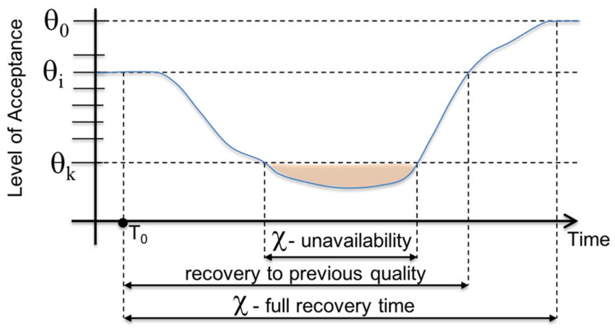


**Fig. 6** Examples of different measures related to the *recovery time*

Indeed, $\zeta(T_{start}, T_{end})$ measures whether ($\zeta(T_{start}, T_{end}) > 0$) or not ($\zeta(T_{start}, T_{end}) = 0$) the system experiences a degradation level greater than $d$ after the occurrence of a disruptive event.

### 4.2.3 Metrics for quantifying the recovery time from failure

According to the discussion in Sect. 4.1, these metrics are related with the *reduced time to recovery* goal, and the *recoverability* view of resilience, and are intended to measure the system ability in quickly recovering from a failure by going back to the original state, or at least to a satisfactory enough state. Broadly speaking, this property thus concerns the system *readiness* for correct service after the occurrence of a disruptive event. Referring to the resilience literature, this property is referred to as *reduced time to recovery* in [12], *"time-based" resilience* in [35], *recovery time* in [19,29].

Within our proposed reasoning framework, we give below examples of possible metrics for this property. In the dependability domain, metrics of this kind are those mainly concerned with the system *availability* (e.g., the mean time to repair (MTTR),[2] or the ratio between the time spent in acceptable states with respect to the total length of the observation interval). We use Fig. 6 to illustrate the metrics we are proposing. In the figure, $\theta_i$ denotes the acceptance criterion (quality level) the system was able to fulfill when the failure occurs.

**Recovery time to a required functionality (unavailability time)** This metric measures the time continuatively spent by the system in unacceptable states because of a disruptive event. Its definition requires the identification of the moment when the system moves to an unacceptable state ($fail\_init$) and the moment when the system returns to an acceptable state ($fail\_end$). Assuming that the disruptive event $e$ occurs at time $T_0$ and $f_1()$ function as defined in (1), we can define the following time instants:

$$fail\_init = \underset{t \in (T_0, \infty)}{\arg\min}\, f_1(\boldsymbol{\sigma}(t)) = 1 \tag{11}$$

[2] MTTR basically corresponds to the mean value of $\chi$ as defined in Eq. (13).

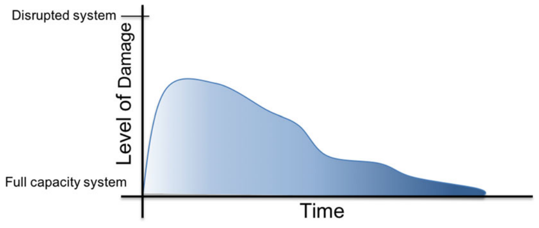


**Fig. 7** Importance of a degradation considering severity of damage and duration in time

and

$$fail\_end = \underset{t \in (fail\_init, \infty)}{\arg\min} \ f_1(\boldsymbol{\sigma}(t)) = 0 \tag{12}$$

Then, the time to recover the system to, at least, an acceptable state can be expressed as:

$$\chi = fail\_end - fail\_init \tag{13}$$

Figure 6 illustrates this time as $\chi - unavailability$. Note that, because of the definition of the $f_1()$ function in (1), this metric only makes sense for events that cause a system degradation beyond some acceptable level. For events that only cause a system degradation that is considered acceptable, it would be $fail\_init = \infty$.

As a special case, if the $f_1()$ function in (1) is defined with $k = 0$ (i.e., such that $f_1(\boldsymbol{\sigma}) = 0$ iff $\boldsymbol{\sigma} \in \theta_0(\boldsymbol{\Sigma})$), then $\chi$ measures the time needed to restore the system to a fully operational state. Figure 6 shows this time as $\chi - Full\ Recovery\ Time$.

**Recovery time to previous quality** When the resilience study assumes that the effect of a disruptive event finishes as soon as the system returns to a quality at least as good as it showed before the event, then a suitable metric for this scenario is:

$$\psi = \underset{t \in (T_0, \infty)}{\arg\min} \{r(\boldsymbol{\sigma}(t)) \leq r(\boldsymbol{\sigma_1})\} \tag{14}$$

where $r()$ is the reward function defined as in Eq. (6) and $\boldsymbol{\sigma_1}$ is the system state when the disruptive event occurred. Figure 6 shows this time as *recovery to previous quality*.

Alternatively, previous recovery times metrics could be slightly modified to measure the time needed to recover the system to some suitable percentage of the fully operational state.

### 4.2.4 Metrics for quantifying a combination of goals

The metrics defined above focuses in isolation on each of the three resilience goals outlined in Sect. 4.1. When a combination of these goals is pursued, some kind of "hybrid" metrics could be more adequate to assess the overall system resilience.

As an example of a possible metric of this kind, we define below a metric that takes into consideration the *consequences from failure* and *recovery time* goals. This metric could be useful when a significant system degradation right after a disruptive event that is quickly recovered is not considered as problematic as a smaller but permanent loss of functionality. Figure 7 graphically depicts such a case, where the curve height represents the "instantaneous" severity of the degradation, while the progressive darkening below the curve represents the increasingly more and more severe damage caused by the system permanence in degraded states. In these cases, we can compute a metric $\omega$ by combining the reward $r()$ in Eq. (6) with the $\psi$ recovery time in (14) as:

$$\omega = \int_{T_0}^{\psi} s(t - T_0)(r(\boldsymbol{\sigma}(t)) - r(\boldsymbol{\sigma}(T_0)))dt \tag{15}$$

where the disruptive event $e$ occurs at time $T_0$ and $s : \Re \rightarrow \Re$ is a monotonically increasing function in $[0 \ldots \infty)$.[3]

## 5 Design strategies and resilience metrics for evolutionary changes

As defined in Sect. 3, *evolutionary changes* denote changes that lead to a modification of the acceptance criterion. These changes can possibly result in a change of the set of acceptable states, where the new set only partially (or even not at all) overlaps with the old one, implying that the system will be in a non-acceptable state for some time. In this section we mainly focus on this type of evolutionary changes, called **EC** *variations* in Sect. 3.2, and discuss strategies and metrics that can be used when we are interested in making a system *flexible*, that is resilient to these variations. As discussed in Sect. 3.2, resilience to the other types of evolutionary changes (*restriction* and *relaxation*) can be instead put in connection with resilience to operational changes, and can thus be dealt with using approaches (strategies and metrics) similar to those presented in Sect. 4.

### 5.1 Resilience strategies

**Resilience as flexibility** The flexibility property is in many aspects similar to other software architecture properties like adaptability, changeability, extensibility and stability, and mainly refers to the system ability to be modified beyond the original design with acceptable effort, so making it ready, for example, to quickly respond to new market conditions [13,28,36,37].

Several strategies have been proposed to achieve this property. For instance, Cossentino et al. [16] define flexibility and extensibility metrics based on classical coupling and cohesion software properties, and hence suggest the adoption of well-established software engineering principles and strategies aimed at achieving the desired level for these properties. Eden and Mens [18] propose a classification of available design paradigms and design patterns well-aligned with the flexibility metrics they define within a reference framework to measure software flexibility. These paradigms and patterns can thus be exploited to achieve some required flexibility level. Achieving high flexibility is also one of the goals of the design patterns and development practices developed within the *software product line* approach [39]. Similarly

[3] Depending on the importance given to the duration of a degradation, function $s(t)$ might be, for example, a logarithmic, linear, polynomial function.

in [36] it is suggested to keep the system design as parametric as possible, both in terms of design parameters and change path enablers. These design choices lead to the definition of possible customization and configuration strategies that depend on the different goals that the systems intend to achieve.

In general, we can say that when strategies to secure flexibility are in place, they require that large parts of a system's behavior change and are verified dynamically. Such radical behavioral changes require a holistic approach that enables online behavior to utilize offline automated development techniques [2]. The practical implications of this approach are, however, not sufficiently studied. Three critical factors underlying its possible application are: *model availability, tool-chain automation*, and *decision-making* criteria and tools. Model-availability and the model-quality are essential for a successful realization of general flexibility. The automated tool-chain and decision making mechanisms such as comparisons and reasoning require that the models, which describe the system, are readily available, accurate, and updated. Flexibility also requires that the tools that work on the models are fully automated and configured in tool-chains that reflects development workflows. An illustration of one such tool-chain is a continuous integration-deployment pipeline. We can conjecture that general flexibility, in some instances, requires updates to the models, tools, and tool-chains in response to radical changes. This meta-adaptation level is currently uncharted territory in research. The final cornerstone in a general flexibility mechanism is support for decision-making. Flexibility requires identification of new acceptable states, generation of correct behavior for the new state-space, and the verification of overall system behavior. This complex process involves several decision types that require support from different reasoning and comparisons strategies for evaluating alternatives, ranking, and decision selection. Some initial research effort in this area inspired by values-based software engineering approaches [10] can be found in [6,24].

## 5.2 Resilience metrics for evolutionary changes

As already pointed out at the beginning of this section, some of the metrics defined in Sect. 4 for resilience assessment with respect to operational changes, can be applied as well for resilience assessment with respect to evolutionary changes. In the following two subsections we first (Sect. 5.2.1) review some of these metrics, highlighting how they can be adapted for flexibility assessment, and then (Sect. 5.2.2) discuss alternative metrics and related issues.

### 5.2.1 Flexibility assessment with metrics from OC scenarios

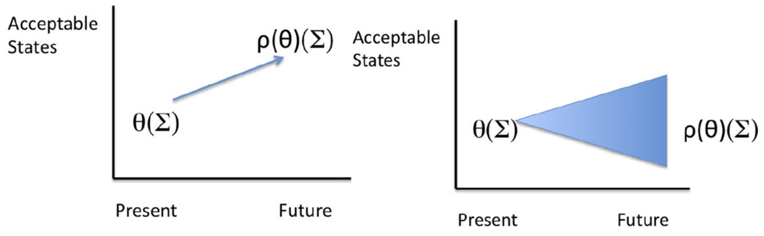
**Metrics for quantifying the ability to prevent failure** In an evolutionary change scenario, a system "failure" occurs when the occurred change makes the current system state no longer acceptable under the new acceptance criterion. A metric analogous to metric (2) (presented in Sect. 4.2.1) could thus be used to assess for how long the system will remain in acceptable states despite the new acceptance spaces resulting from the occurrence of evolutionary changes. Note that if all changes are of *relaxation* type, then the system will surely remain acceptable along all the observation interval, while this cannot be guaranteed in case of *restriction* or *variation* changes.

**Metrics for quantifying the recovery time from failure** An evolutionary change of *restriction* or *variation* type may move the current state of the system from being well accepted to the survival space. As an example scenario, we can imagine that some new regulation is introduced concerning data confidentiality. An IT data management system obviously continues to keep its operating capabilities as before the change, but they could be no longer compliant with the new rules. Resilience metrics as those presented in Sect. 4.2.3 could thus be used to measure the minimum time necessary to bring the system within the new acceptance space.

### 5.2.2 Flexibility assessment under deep uncertainty scenarios

Tackling evolutionary changes often implies to consider a wider temporal horizon than the one related to operational changes. Therefore, it is more likely that the changes to be faced are of the *unexpected* type. As a consequence, defining metrics for the assessment of resilience in these scenarios requires to thoroughly take into account issues related with uncertain future.

Over the past several years, researchers have started studying the notion of uncertainty in modelling and analyzing complex systems and the existence of different types of uncertainty is now widely recognized [4,20,32,34,41,43]. The literature provides several definitions of uncertainty, most of them classify uncertainties according to their level (from determinism to complete ignorance), nature (aleatory or epistemic), and source (model structure, data and parameters, but also changes in the operational environment, dynamics in the availability of resources, and variations of user goals) [32,34,42]. One challenge in this context is being able to identify/recognize the presence of uncertainty in a given system. Some proposals can be found on this theme. For example, [31] proposes a methodology that guides the software engineers in recognizing the existence of uncertainty. A bayesian approach has been proposed in [8] to evaluate the presence of uncertainty (called surprise) using a metric that measures the distance between the prior and the posterior probability distributions. Once the presence of some type of uncertainty has been recognized, dealing with it can be associated to three different paradigms that can be adopted to model the future [26]. In the first one, (i) the idea is to *anticipate the future based on best available knowledge* with the implicit assumption that knowledge can be improved by data collection and surprises can be included altering the original model. This corresponds to the idea of a clear enough or deterministic future [42]. In the second paradigm, (ii) *the future is treated as quantifiable* with (combination of) probability distribution and study of the uncertainty propagation. This corresponds to a level of uncertainty characterised as "statistical" or probabilistic [42]. The third one (iii) explores *multiple possible futures* considering different possible scenarios. This corresponds to a level of uncertainty characterised as "unknown future" [42]. Following this classification, we can distinguish several ways in which flexibility can be reached. Figure 3 illustrates the system state space in case of evolutionary changes. However, there are different ways in which evolutionary changes can lead to the definition of the new set of acceptable states. Figures 8a, b



**(a)** Flexibility by anticipating the future based on best available knowledge **(b)** Flexibility when the future is quantifiable

**Fig. 8** Flexibility with modelling paradigms (i) and (ii), adapted from [26]

and 9a illustrate how, starting from a set of acceptable states $\theta(\boldsymbol{\Sigma})$ a new future set of acceptable system states can be reached according to paradigms (i), (ii) and (iii) for modelling the future. Figure 9b, instead, shows how it is possible to combine the three paradigms to address different sources of uncertainty within a problem.

Given the definition of variation in Sect. 3, with the assumption that $\rho(\theta_k) \subseteq \theta_s$, we can define the following function $dist : \boldsymbol{\Theta} \times \boldsymbol{\Theta} \rightarrow \Re$ as a possible measure of the distance between $\theta_k$ and $\rho(\theta_k)$ (but other distance functions could be defined):

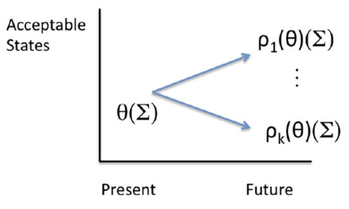
$$dist(\theta_k, \rho(\theta_k)) = inf\ \{dist(x, y) | x \in \theta_k, y \in \rho(\theta_k)\} \tag{16}$$

With this definition, $dist(\theta_k, \rho(\theta_k)) = 0$ if $\theta_k$ and $\rho(\theta_k)$ at least partially overlap, while it is increasingly greater than zero as the separation between the two sets increases. By expressing each point $x$ and $y$ in a cartesian coordinates space, the function $dist$ can be computed for each modelling paradigm. Specifically, for (i) $dist$ is as defined before, while for paradigm (ii) the value obtained with $dist$ will be weighted by the confidence interval obtained by the adopted probability distribution. For paradigm (iii), the function $dist$ will be evaluated for each $\rho(\theta_k)(\boldsymbol{\Sigma})$. Then, the combination of the different paradigms will be characterized by the evaluation of $dist$ for every $\rho(\theta_k)(\boldsymbol{\Sigma})$, and each of them will be weighted by the confidence interval obtained by the adopted probability distribution.

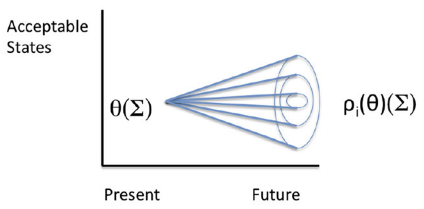
Using the $dist()$ function as a basis for a flexibility metric, we can characterize how far it can be the new set of acceptable states $\rho(\theta_k)$ that the system is able to reach from its current state $\theta_k$, in case of evolutionary changes. The further this set, the greater the system flexibility is. We point out that this kind of metric is independent of the change that could actually occur in the future, as it only measure a sort of system "streachability" property along many possible directions. Thus, it seems well suited to scenarios characterized by deep uncertainty about the changes that could actually occur.

With these definitions, the goal of a flexible architecture could be stated as: **maximize** $dist(\theta_k, \rho(\theta_k))$.

Other different approaches can be adopted as well to deal with the uncertainty about future evolutionary changes [6,24], based on the concepts in [10]. For example, following the approach introduced by Letier et al. [24], it is possible to exploit the concept of *value of information* to support the decision making process when different options



**(a)** Flexibility by exploring multiple scenarios



**(b)** Flexibility by combining paradigms (i), (ii) and (iii)

**Fig. 9** Flexibility with different modelling paradigms, adapted from [26]

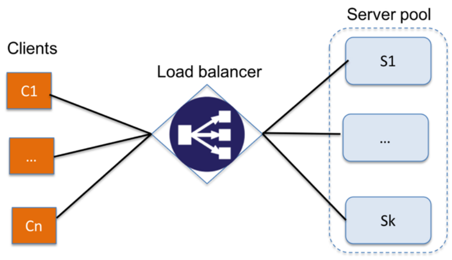
are available, concerning for example possible design decisions. More specifically, to each evolutionary change introducing a variation of the acceptance criterion it can be associated a metric like the *expected value of information* [24], which evaluates the expected gain in terms of net benefit related to that change with and without additional information. This metric is complemented with the evaluation of the *Risk* associated to the change, evaluated by the loss probability and the probable loss magnitude. In this context, the goal of a flexible system can be stated as: **minimize the *Risk* associated to each change of the acceptance criterion.**

Alternatively, another possible metric that could be applied is defined in [6], based on the *real options* theory developed in the financial domain [27]. An option represents a choice regarding an investment opportunity (without obligations) under given terms for a fixed period of time in the future for a tangible (real) asset. A parallel is built considering a likely future change in a system analogous to buying an option on an asset, with a price corresponding to the cost of implementing the change. The method in [6] defines a way to value the flexibility of the system to accommodate the change taking into account parameters like effort, schedule, budget and so on. This metric can be used to associate an *economical value* to each evolutionary change, and the goal of a flexible system is to be able to minimize it.

## 6 Case study

In this section we illustrate some of the main concepts introduced in the previous sections, using to this end examples based on the ZNN.COM case study [15], showing possible mutual relationships among the different types of changes (OC and EC) and the various types of resilience discussed so far.

ZNN.COM (Fig. 10a) reproduces a news system that delivers multimedia (static and dynamic) content to its customers. It adopts a web-based client-server architecture, where a dynamic pool of replicated servers receives content requests from a set of clients; a load balancer balances the load among servers. One of the main quality indexes for this system is its response time to client requests, which depends on factors like the number of servers in the pool, the load generated by clients, the bandwidth and latency of network connections, and the fidelity of the delivered content with respect to the original one stored into the system. Another relevant quality index is the system cost, which depends on the number of used servers.



**(a)**

| attribute | domain |
|---|---|
| $c$.EXPRSPTIME | $\mathbb{N}^+$ |
| $s$.COST | $\mathbb{N}^+$ |
| $s$.LOAD | $\mathbb{N}^+$ |
| $s$.FIDELITY | $\{full, reduced\}$ |
| $s$.ACTIVE | $\{yes, no\}$ |
| $h$.BANDWIDTH | $\mathbb{N}^+$ |
| $h$.LATENCY | $\mathbb{N}^+$ |
| $p$.LOAD | $\mathbb{N}^+$ |

**(b)**

**Fig. 10** ZNN.COM case study

The table in Fig. 10b shows attributes that can concur to the definition of the ZNN.COM system state space $\boldsymbol{\Sigma}$, for each client $c$, server $s$, network connection $h$ and load balancer $p$ in the system. Besides the attribute names, the table also shows their domain, where we assume a suitable discretization for real-valued attributes (see [14]).

Given this state space, a possible state-based acceptance criterion could be defined as:

$$\theta_0 = \forall c : c.\text{EXPRSPTIME} < \text{MAXRSPTIME}_0 \wedge \forall s : s.\text{FIDELITY} = full \tag{17}$$

where $\text{MAXRSPTIME}_0$ is a threshold on the experienced response time.

The occurrence of operational changes (OC) affecting the system itself or its environment could impair the system ability to remain within the set $\theta_0(\boldsymbol{\Sigma})$. As an example of these changes, we consider load variations. To this end, we assume that possible load values are classified into three consecutive intervals: $I_0 = [0 \ldots l_0]$ (*normal load*), $I_1 = [l_0 \ldots l_1]$ (*high load*), $I_2 = [l_1 \ldots +\infty]$ (*extreme load*), and that the minimum number of servers needed to fulfill $\theta_0$ under a normal or high load is $N_0$ or $N_1$, respectively, with $N_0 < N_1$. The ZNN.COM system designers could thus face some possible different situations.

*Situation 1: $\theta_0$ is a strict requirement* In this case no other acceptable states exist beyond $\theta_0(\boldsymbol{\Sigma})$. According to the discussion in Sect. 3.2, the system should thus be made *robust* with respect to any possible load value. This property can be achieved by introducing into the system design redundancy techniques (Sect. 4.1), based for example on the statical overprovisioning of the server pool, or (probably better from a cost viewpoint) on the proactive scaling-in/out of the number of servers in anticipation of foreseen load variations. However, cost considerations would probably limit in both cases the maximum number of servers in the pool, for instance equal to $N_1$. In this case, the system is robust with respect to $\theta_0$ and load variations in the $I_0$ and $I_1$ ranges. For loads in the $I_2$ range the system is not able to provide acceptable performance; however, it can return to be acceptable as soon as the load decreases below the $l_1$ threshold. Hence, the survival space with respect to load variations is $\theta_s(\boldsymbol{\Sigma}) = \boldsymbol{\Sigma} \backslash \theta_0(\boldsymbol{\Sigma})$ and the system is *recoverable* with respect to any load variation.

The effectiveness of a given solution (e.g. static redundancy or dynamic scaling-in/out) can be analyzed with respect to the various perspectives and corresponding metrics discussed in Sect. 4.2. For instance, the actual system robustness (i.e., ability to continuously remain within the set $\theta_0(\boldsymbol{\Sigma})$) can be assessed using metrics (2) or (4), metric (14) could be used to assess how quickly the system is able to return to an acceptable state after a peak load that deteriorates its performance, and metric (15) could be used to assess the consequences of the deterioration.

*Situation 2: $\theta_0$ is a relaxable requirement* The ZNN.COM system designers could realize that the incurred costs to make the system robust with respect to load variations in the $I_0$ and $I_1$ ranges are too high. A new elicitation phase for system requirements could then lead to realize that system users highly appreciate its responsiveness, while they can accept a temporary content fidelity reduction (e.g. text-only instead of multimedia news) that, from the system viewpoint, allows to process a given load using less computing power. This leads to an evolutionary change of the system requirements based on the following *relaxation* of $\theta_0$:

$$\theta_1 = \forall c : c.\textsc{expRspTime} < \textsc{MaxRspTime}_0 \tag{18}$$

Then, after realizing that some users of ZNN.COM are willing to accept a less responsive system against a compensation in their monthly fee, a further relaxation could be introduced:

$$\theta_2 = \forall c : c.\textsc{expRspTime} < \textsc{MaxRspTime}_1 \tag{19}$$

with $\textsc{MaxRspTime}_0 < \textsc{MaxRspTime}_1$. These progressively relaxed acceptance criteria allow to design the ZNN.COM system as a *gracefully degradable* system, where states in $\theta_0(\boldsymbol{\Sigma})$ make all its users fully satisfied, states in $\theta_1(\boldsymbol{\Sigma})\backslash\theta_0(\boldsymbol{\Sigma})$ reduce the degree of satisfaction of all users, while states in $\theta_2(\boldsymbol{\Sigma})\backslash\theta_1(\boldsymbol{\Sigma})$ partially satisfy only a subset of the users. States in $\boldsymbol{\Sigma}\backslash\theta_2(\boldsymbol{\Sigma})$ are not acceptable for any user, but the system can recover from them after a suitable reduction of the system load. Operationally, the ZNN.COM system could be managed according to some reactive self-adaptation technique, that adjust the number of servers and the delivered content fidelity in response to load variations, according to some degradation policy (Sect. 4.1). The effectiveness of the adopted policy in terms of tradeoff between cost reduction and users satisfaction could be assessed using performability metrics (7), (8), (10) in Sect. 4.2.

*Situation 3: Acceptance criteria variation* After the ZNN.COM news system has been designed as a gracefully degradable system according to the progressively more relaxed acceptance criteria $\theta_0$, $\theta_1$ and $\theta_2$, the demand emerges of building a specialized ZNN4ARTIST.COM version for the artists community. Requirements elicitation for this new version reveals that its prospective users do not care too much about system responsiveness, but strictly require full quality content delivery. The corresponding acceptance criterion could thus be stated as:

$$\theta_3 = \forall s : s.\textsc{fidelity} = full \tag{20}$$

$\theta_2(\boldsymbol{\Sigma})$ and $\theta_3(\boldsymbol{\Sigma})$ only partially overlap with each other; hence, $\theta_3$ can be seen as the result of a *variation* EC with respect to $\theta_2$, caused by a modification of the user preferences: states considered acceptable under $\theta_2$, albeit in degraded mode (e.g., states where $s.fidelity = reduced$), are no longer acceptable at all under $\theta_3$, which instead allows to accept states not acceptable under $\theta_2$ (e.g., states where s.EXPRSPTIME > MAXRSPTIME$_1$). Hence, the designers of ZNN4ARTIST.COM must quite deeply re-think the $\theta_2$-based graceful degradation policy embedded in the managing system of ZNN.COM. As discussed in Sect. 5.2.1, metrics like those presented in Sect. 4.2.3 could be used to measure the effort required to this end.

## 7 Discussion and conclusions

### 7.1 Discussion

As stated in the introductory part, our goal has been to contribute to the definition of the fundamental pillars, summarized in Fig. 1, of a conceptual framework that should underpin any concrete effort towards resilient systems design and development.

We remark that our framework aims at embodying the big picture of resilient systems design, and does not go into details of specific aspects, which are instead investigated by more narrowly focused papers in literature. Hereafter, we discuss some issues concerning our contribution.

The concept of *acceptance criteria* $\theta_i$ introduced in Sect. 3 plays a relevant role in the definition of our conceptual framework. Indeed, it provides the basis for the idea of partitioning the system state space in subsets representing different degrees of acceptability, and the consequent definition of the different facets of resilience in terms of trajectories over these subsets, and of assessment metrics in terms of functions over sets of states or state trajectories. Each $\theta_i$ is actually an abstract representation of a set of requirements with respect to which we can assess the resilience of a system. We do not give details about how these sets of requirements are elicited and specified. Suggestions about how this abstract concept can be detailed and reified can be found in work concerning requirements specification for self-adaptive systems (e.g. [7,40,44]). Indeed, even if self-adaptation is not the only strategy to achieve resilience to changes (for example, in case of systems using static redundancy), it is undoubtedly one of the most relevant. For example, our abstract idea of progressively looser acceptance criteria $\theta_i$, $i = 0, 1, 2, \ldots$ can be mapped to the concept of requirements relaxation in [44], where a fuzzy logic-based formal semantics is given for this type of evolutionary change. Different, more goal-oriented ways for specifying "relaxed" requirements are proposed in [7,40]), together with suitable formal semantics (based on $OCL_{TM}$ and fuzzy logic, respectively).

The discussion in Sect. 3 mostly assumes a linear ordering among the $\theta_i$'s, corresponding to the containment relationships of the sets $\theta_i(\boldsymbol{\Sigma})$'s. As evidenced above, this can be seen as the result of the progressive application of *relaxation* (or, in the opposite direction, *restriction*) EC to the initial acceptance criterion $\theta_0$. In general, one could think of relaxation EC applied to different parts of the whole set of conditions (requirements) included in $\theta_0$. This would result in different chains of progressively looser requirements, actually forming a tree rooted at $\theta_0$. An example of this can be found in the case study of Sect. 6, where $\theta_3$, rather than a variation of $\theta_2$, could also be seen as a relaxation of $\theta_0$ along a different direction with respect to the chain $\theta_0, \theta_1, \theta_2$.

In our opinion, this "tree" of acceptance criteria can be mainly seen as the result of different specializations of a system for different domains, characterized by partially overlapping set of requirements. As a consequence, our notion of *flexibility* seems to us well suited to embrace resilience to this kind of scenarios, as it focuses on the system ability to face *variations* in the set of requirements it should fulfill.

We also point out that our conceptual framework refers to intuitive concepts of states, state trajectories, and metrics based on them. We do not have purposely given a formal semantics for these concepts, to not get stuck with the details of a specific formalism and its possible limits. However, we are aware that, in this respect, an important related body of work exist, concerning the use of formal theories to give a formally sound semantics to our intuitive concepts, e.g. one of the different flavors of temporal logics [1,21].

The metrics for resilience assessment we have presented in previous sections, in particular in Sects. 4.2 and 5.2, are likely to be evaluated under a probabilistic setting, based on assumptions about a set of possible events that can modify the system or environment state and their corresponding occurrence probabilities. From these underlying assumptions, the moments or probability distribution of those metrics can be evaluated, thus obtaining indications about the actual system resilience or suggestions about possible interventions for its improvement.

However, we must consider that these assumptions are always based on some kind of limited knowledge that can lead us to imprecisely evaluate the occurrence probability of some events, or the impact they could have, or to ignore at all events that could instead actually occur. In other words, the resilience we assess through those metrics is based on the knowledge we currently have about the system and its environment.

This resilience, and the knowledge underlying it, can be challenged by the occurrence of *unexpected* events, as discussed in Sect. 3.3, where what is not expected can be the event itself, or its consequences. Indeed, these events, as remarked in [46,47] where they are named "surprises", can lead the system to touch or surpass the finite envelope that always bounds its resilience, because of intrinsic limitations in the available resources, or because of the limited knowledge of its designers. It is still possible to take advantage of these unexpected events that challenge the system resilience. For example, by updating the knowledge that is used to assess system resilience, or by applying an evolutionary change that updates the system requirements and avoids damage upon the occurrence of the same type of event, or by evolving the system functionality—a case where the area of study of *antifragile* systems [33] could be applied. An interesting proposal to formalize a concept of surprise within a Bayesian framework can be found in [8], where a surprise is defined in terms of the distance between prior and posterior probability distributions. A thorough assessment of the system resilience must thus include the assessment of the system readiness to face these surprises. This is particularly challenging, as by definition we have very limited, if not at all, knowledge about them. Metrics like those discussed in Sect. 5.2.1 aim at assessing this kind of readiness, thus constituting in our opinion one of the most interesting avenues of research in the field of system resilience.

## 7.2 Future work

As already discussed, we reported in this paper our first results towards our ultimate research objective of defining a methodology for systematic design, generation, and validation of resilient, software-intensive, socio-technical systems with assurances. Our future work will follow two separate and yet interleaved paths.

On the one hand, we intend to focus on resilience assessment with respect to evolutionary changes, as it is linked to the unexpected events issues discussed above, and appears less mature than resilience assessment with respect to operational changes. From our viewpoint, assessing the systems resilience when facing evolutionary changes involves an analysis of the uncertainty about predicting a distant future, including how much the system engineers know about the alternatives in the future, their likelihood, and their properties. This research line deserves further investigation that we plan to pursue also taking advantages from the results coming from different disciplines with more mature understanding of the topic, like economics and environmental modelling.

On the other hand, we plan to investigate how the proposed conceptual framework can be exploited to provide support for architectural reasoning and assurances for the resilience property, taking into account different specialty areas that contribute to resilience. A further step forward will consist in the definition of a resilience-based development process that supports the complete system life-cycle, including online adaptation and evolution.

**Acknowledgements** This work has been partially supported by the SISMA Italian national research project (MIUR, PRIN 2017, Contract 201752ENYB) and by ALADINO project (funded by KKS Sweden, n° 20200117).

**Funding** Open access funding provided by Politecnico di Milano within the CRUI-CARE Agreement.

## References

1. Agha G, Palmskog K (2018) A survey of statistical model checking. ACM Trans Model Comput Simul. https://doi.org/10.1145/3158668
2. Andersson J, Baresi L, Bencomo N, de Lemos R, Gorla A, Inverardi P, Vogel T (2013) Software engineering processes for self-adaptive systems. Springer, Berlin, pp 51–75
3. Annarelli A, Nonino F (2016) Strategic and operational management of organizational resilience: current state of research and future directions. Omega 62(C):1–18
4. Ascough J II, Maier H, Ravalico J, Strudley M (2008) Future research challenges for incorporation of uncertainty in environmental and ecological decision-making. Ecol Model 219(34):383–399
5. Avizienis A, Laprie JC, Randell B, Landwehr CE (2004) Basic concepts and taxonomy of dependable and secure computing. IEEE Trans Dependable Sec Comput 1(1):11–33

6. Bahsoon R, Emmerich W (2004) Evaluating architectural stability with real options theory. In: 20th international conference on software maintenance (ICSM 2004), 11–17 September 2004, Chicago, IL, USA. IEEE Computer Society, pp 443–447
7. Baresi L, Pasquale L, Spoletini P. Fuzzy goals for requirements-driven adaptation. In: 2010 18th IEEE international requirements engineering conference, pp 125–134
8. Bencomo N, Belaggoun A (2014) A world full of surprises: Bayesian theory of surprise to quantify degrees of uncertainty. In: Jalote P, Briand LC, van der Hoek A (eds) 36th international conference on software engineering, ICSE'14, companion proceedings, Hyderabad, India, May 31–June 07, 2014. ACM, pp 460–463. https://doi.org/10.1145/2591062.2591118
9. Bergstrom J, van Winsen R, Henriqson E (2015) On the rationale of resilience in the domain of safety: a literature review. Reliab Eng Syst Saf 141:131–141 **(Special Issue on Resilience Engineering)**
10. Boehm B, Sullivan K (2000) Software economics: a roadmap. In: Proceedings of the Conference on The Future of Software Engineering (ICSE '00). ACM, New York, p 319–343. https://doi.org/10.1145/336512.336584
11. Braithwaite J, Wears R, Hollnagel E (2015) Resilient health care: turning patient safety on its head. Int J Qual Health Care 27:418–420
12. Bruneau M, Chang SE, Eguchi RT, Lee GC, Rourke TDO, Reinhorn AM, Shinozuka M, Tierney K, Wallace WA, von Winterfeldt D (2003) A framework to quantitatively assess and enhance the seismic resilience of communities. Earthq Spectra 19(4):733–752
13. Byrd TA, Turner DE (2000) Measuring the flexibility of information technology infrastructure: exploratory analysis of a construct. J Manag Inf Syst 17(1):167–208
14. Cámara J, de Lemos R, Vieira M, Almeida R, Ventura R (2013) Architecture-based resilience evaluation for self-adaptive systems. Computing 95(8):689–722
15. Cheng S, Garlan D, Schmerl BR (2009) Evaluating the effectiveness of the rainbow self-adaptive system. In: 2009 ICSE workshop on software engineering for adaptive and self-managing systems, SEAMS 2009, Vancouver, BC, Canada, May 18–19, 2009. IEEE Computer Society, pp 132–141
16. Cossentino M, Lodato C, Lopes S, Ribino P, Palermo V (2015) Metrics for evaluating modularity and extensibility in HMAS systems. In: Proceedings of the 2015 international conference on autonomous agents and multiagent systems, pp 1061–1069
17. Di Marzo Serugendo G (2009) Robustness and dependability of self-organizing systems—a safety engineering perspective. In: Guerraoui R, Petit F (eds) Stabilization, safety, and security of distributed systems. Springer, Berlin, pp 254–268
18. Eden A, Mens T (2006) Measuring software flexibility. IEE Proc Softw 153:113–125
19. Erol O, Henry D, Sauser B, Mansouri M (2010) Perspectives on measuring enterprise resilience. In: 2010 IEEE international systems conference, pp 587–592
20. Giese H, Bencomo N, Pasquale L, Ramirez AJ, Inverardi P, Wätzoldt S, Clarke S (2014) Models@run.time: foundations, applications, and roadmaps, chap. Living with uncertainty in the age of runtime models. Springer, pp 47–100
21. Konur S (2013) A survey on temporal logics for specifying and verifying real-time systems. Front Comput Sci 7(3):370–403
22. Krueger CW (1992) Software reuse. ACM Comput Surv 24(2):131–183
23. Laprie JC (2008) From dependability to resilience. In: DSN 2008
24. Letier E, Stefan D, Barr ET (2014) Uncertainty, risk, and information value in software requirements and architecture. In: 36th international conference on software engineering, ICSE'14, Hyderabad, India, May 31–June 07, 2014. ACM, pp 883–894
25. Littlewood B, Strigini L (2000) Software reliability and dependability: a roadmap. In: Proceedings of the conference on the future of software engineering, pp 175–188
26. Maier H, Guillaume J, van Delden H, Riddell G, Haasnoot M, Kwakkel J (2016) An uncertain future, deep uncertainty, scenarios, robustness and adaptation: how do they fit together? Environ Model Softw 81:154–164
27. Myers SC (1984) Finance theory and financial strategy. Interfaces 14(1):126–137
28. Naab M, Stammel J (2012) Architectural flexibility in a software-system's life-cycle: systematic construction and exploitation of flexibility. In: Proceedings of the 8th international ACM SIGSOFT conference on Quality of Software Architectures, QoSA, pp 13–22
29. Najarian M, Lim GJ. Design and assessment methodology for system resilience metrics. Risk Analysis. Online first. https://onlinelibrary.wiley.com/doi/abs/10.1111/risa.13274

30. Patriarca R, Bergstrom J, Gravio GD, Costantino F (2018) Resilience engineering: current status of the research and future challenges. Saf Sci 102:79–100
31. Perez-Palacin D, Mirandola R (2014) Dealing with uncertainties in the performance modelling of software systems. ACM, pp 33–42. https://doi.org/10.1145/2602576.2602582
32. Perez-Palacin D, Mirandola R (2014) Uncertainties in the modeling of self-adaptive systems: a taxonomy and an example of availability evaluation. ACM, pp 3–14. https://doi.org/10.1145/2568088.2568095
33. Ramezani J, Camarinha-Matos LM (2020) Approaches for resilience and antifragility in collaborative business ecosystems. Technol Forecast Soc Change 151:119846. https://doi.org/10.1016/j.techfore.2019.119846
34. Ramirez AJ, Jensen AC, Cheng BHC (2012) A taxonomy of uncertainty for dynamically adaptive systems. In: Proceedings of the 7th international symposium on software engineering for adaptive and self-managing systems, SEAMS'12. IEEE Press, Piscataway, NJ, USA, pp 99–108
35. Rose A, Liao SY (2005) Modeling regional economic resilience to disasters: a computable general equilibrium analysis of water service disruptions*. J Reg Sci 45(1):75–112
36. Ross AM, Rhodes DH, Hastings DE (2008) Defining changeability: reconciling flexibility, adaptability, scalability, modifiability, and robustness for maintaining system lifecycle value. Syst Eng 11(3):246–262
37. Salama M (2018) Architectural stability of self-adaptive software systems. Ph.D. thesis, University of Birmingham, UK
38. Schmeck H, Müller-Schloer C, Çakar E, Mnif M, Richter U (2010) Adaptivity and self-organization in organic computing systems. ACM Trans Auton Adapt Syst 5(3):10:1–10:32
39. Software Product Lines (2001) Practices and patterns. Addison-Wesley Longman Publishing Co., Inc., Boston
40. Souza VES, Lapouchnian A, Robinson WN, Mylopoulos J (2011) Awareness requirements for adaptive systems. In: Proceedings of the 6th international symposium on software engineering for adaptive and self-managing systems, SEAMS 2011. ACM, New York, NY, USA, pp 60–69
41. van Asselt MBA, Rotmans J (2002) Uncertainty in integrated assessment modelling. Clim Change 54(1):75–105. https://doi.org/10.1023/A:1015783803445
42. Walker W, Harremoes P, Romans J, van der Sluus J, van Asselt M, Janssen P, Krauss M (2003) Defining uncertainty. A conceptual basis for uncertainty management in model-based decision support. Integr Assess 4(1):5–17
43. Weyns D, et al (2013) Perpetual assurances for self-adaptive systems. In: de Lemos R, Garlan D, Ghezzi C, Giese H (eds) Software engineering for self-adaptive systems III. Assurances - international seminar, Dagstuhl Castle, Germany, December 15–19, 2013, Revised Selected and Invited Papers. Lecture notes in computer science, vol 9640. Springer, pp 31–63
44. Whittle J, Sawyer P, Bencomo N, Cheng BH, Bruel JM (2010) Relax: a language to address uncertainty in self-adaptive systems requirement. Requir Eng 15(2):177–196
45. Wiig S, Fahlbruch B (2019) Exploring resilience: a scientific journey from practice to theory. Springer International Publishing, Cham
46. Woods DD (2015) Four concepts for resilience and the implications for the future of resilience engineering. Reliab Eng Syst Saf 141:5–9 **(Special Issue on Resilience Engineering)**
47. Woods DD (2019) Essentials of resilience, revisited. In: Handbook on resilience of socio-technical systems, chap. 4. Edward Elgar Publishing, pp 52–65. https://EconPapers.repec.org/RePEc:elg:eechap:17780_4