Safety vs. Sustainability Design: Analogies, Differences and Potential Synergies

Guillermo Rodriguez-Navas
Mälardalen University
Västeras, Sweden
guillermo.rodriguez-navas@mdh.se

Leticia Duboc
State University of Rio de Janeiro
Rio de Janeiro, Brazil
leticia@ime.uerj.br

Stefanie Betz
Karlsruhe Institute for Technology
Karlsruhe, Germany
stefanie.betz@kit.edu

Ruzanna Chitchyan
University of Leicester
Leicester, UK
rc256@leicester.ac.uk

Birgit Penzenstadler
California State University Long Beach
California, USA
birgit.penzenstadler@csulb.edu

Colin C. Venters
University of Huddersfield
Huddersfield, UK
c.venters@hud.ac.uk

***Abstract*—The idea that there are important parallels between safety and sustainability and that software engineers might be able to take lessons learned from safety and apply them to sustainability has been voiced and initially explored before. This paper extends the analysis of similarities, differences, and potential synergies between the two concepts, according to four different dimensions of these domains: systemicity, complexity, certification and social perception.**

***Index Terms*—Sustainability, safety, non-functional requirements.**

I. Introduction

Driven by the recent increased interest on sustainability topics, software engineering researchers have turned to study the methods and techniques that could facilitate sustainability engineering in and through software systems. Some authors have pointed out that safety shares many common aspects with sustainability, and that system designers might be able to take lessons learned from safety, and apply them to sustainability [1]. But up to date only very preliminary analyses have been performed, which shedded little light on how to achieve this goal.

Safety engineering is a very mature discipline. It originated in the field of process control and spread towards the field of computer systems back in the 80's, when digital control started to be used for critical applications; see e.g. [2]. In last decades, the safety aspects of software have received even more attention, because of the ubiquity of computer systems in almost every aspect of our daily life. Any software-based systems used in a critical application, such as transportation, health-care or energy systems, must nowadays adhere to strict safety regulations and best practices; otherwise, these products cannot be certified. Many companies share and openly promote this concern for safety, as part of their corporate image. Thus, one could say that safety, and specifically *software safety*, has already reached some of the goals that sustainability design is still defining, and can be a source of inspiration for this domain.

Yet, today there is no clear understanding of how exactly the safety and sustainability domains can be related. Disparate views can be found in the literature: whereas some authors report that "safety must be traded for increased sustainability" (in the context of vehicle design [3]), others claim that "safety can bring more sustainability" (in the context of intelligent transportation systems [4]). Others advocate the integration of safety with systems thinking, by introducing sociotechnical aspects [5] that can help to handle the complexity of some of the current safety-critical systems.

The goal of this paper is to critically review the similarities, differences, and potential synergies between these two domains, with special emphasis on the early analysis and design phases. This work is the result of an open on-going dialogue between researchers on sustainability design and software safety design, which was sparked by the elaboration of the *Karlskrona Manifesto for Sustainability Design* [6]. Some questions we wish to answer are: what sustainability can learn from safety? Which aspects of safety are similar and useful for sustainability design? Which, if any, of the techniques for safety design can be applicable for sustainability? In which respects does the domain of sustainability differ from safety? What extensions/additions/changes does handling such differentiation require for sustainability design?

Even if the answers to these questions are interesting to software engineers in general, they can be particularly useful to the requirements engineers. Requirements engineering helps to identify the relevant stakeholders and to shape the scope and purpose of the system. If these are made without consideration for sustainability, the rest of the software development activities, including verification, are unlikely to produce systems that will support this important concern.

Finally, although the relevant literature has been reviewed (in Section 3 below), our intention is not to provide a systematic literature survey of the intersection between these two areas, but instead to provide a new understanding of their relationship. This paper is organized as follows: it introduces the basic concepts of safety and sustainability in Section 2, followed by a summary of existing works comparing both concepts, in Section 3. Section 4 discusses the parallels between safety and sustainability, while Section 5 summaries the paper and presents our conclusions.

II. Background

Before comparing safety and sustainability and their relationships with software engineering, we will briefly review the fundamental notions of these two domains.

A. Safety

Safety is typically defined as freedom from accidents or losses [7]. Thus, safety aims at a methodology avoiding present and potential losses which can be caused by unexpected events (called accidents) or poor (unreliable) service of the system.

Traditionally, an accident is seen as an undesired and unplanned (but not necessarily unexpected) chain of events that results in loss. Sometimes this chain of events does not cause a loss at a given time and under specific conditions, but that loss could potentially be incurred under different conditions; such events are called "near miss" or "incident".

To manage the avoidance of accidents, safety engineering looks for their possible causes (called hazards) and assesses the probability of an accident along with the seriousness of resultant losses (called risk). A hazard is a state or set of conditions of a system (or an object) that, together with other conditions in the environment of the system (or object), will lead inevitably to an accident (loss event). Thereby, a hazard is defined with respect to the environment of the system or component, and what constitutes a hazard depends upon where the boundaries of the system are drawn.

Risk is the severity of a hazard combined with (1) the likelihood of the hazard leading to an accident (sometimes called danger) and (2) hazard exposure or duration (sometimes called latency) [7]. A common way to assign risk is through Safety Integrity Level (SIL), but other measures exist, depending of the adopted safety standard [8].

Reliability is the probability that a piece of equipment or component performs its intended function for a prescribed time and under stipulated environmental conditions. While dependability is concerned with the incidence of failures and aims at increasing reliability; safety is concerned with the occurrence of accidents or mishaps [9]. Where system failures are defined in terms of system services, safety is defined in terms of external consequences. If the required system services are specified incorrectly, then a system may be unsafe, though perfectly reliable. Conversely, it is feasible for a system to be safe, but unreliable. Enhancing the reliability of software components, though desirable and perhaps necessary, is not sufficient to ensure that they will not contribute to a mishap [7].

Hazard analysis is the process of identifying the different types of hazards a system may experience and their causes. Hazard analysis is performed at several different stages in the design lifecycle (e.g., preliminary, subsystem, system, and operational hazard analysis), and there are a number of supporting methodologies (e.g., hazard and operability studies, or HAZOPS, fault tree analysis, or FTA, and failure modes and effects analysis, or FMEA [7], [5].

The basic idea in safety is to focus on the consequences that must be avoided rather than on the requirements of the system itself (since those might be the very source of undesired consequences). Next, because the occurrence or nonoccurrence of a mishap may depend on circumstances beyond the control of the system under consideration, attention is focused on preventing hazards, which are conditions (i.e., states of the controlled system) that can lead to a mishap, rather than preventing mishaps directly.

Different techniques and strategies can be applied in order to eliminate or mitigate system hazards. Regarding the design of system safety measures, a set of five principles has been defined in [10], which can be considered general and thus applicable to software.

1) The fail-safe principle. This principle requires that the failure of a component results in an operational state that cannot contribute to the chain of events potentially causing an accident. This principle is not always realizable, but should be considered for every component.
2) The safety margins principle states that the "safe" operational state must be defined with certain distance (or tolerance) from the hazard threshold, so as to accomodate uncertainties.
3) The un-graduated response principle recommends to not apply countermeasures progressively, but to try the most aggressive approach first, in order to block the chain of events potentially leading to a hazard. This can be seen as a reinterpretation of the rules of *intrinsically safe design*.
4) The defense-in-depth principle is a central principle for safety design, which recommends to define a multiplicity of safety measures, of diverse nature, and acting on different system levels.
5) The observability-in-depth principle requires the system design to provide means to monitor the faulty/non-faulty state of the component, so the user cannot have a false sense of safety due to lack of information.

These principles are related to the notions of hazard and the existence of a chain of event leading to it. Some work has criticized the rigidity of this approach, advocating the use of feedback-loop control in order to identify and act upon hazardous states [11].

Complementary to the recommendations for the design of safety measures, some researchers have focused on the provision of adequate *safety assurance*. Assurance cases are documented bodies of evidence that provide valid and convincing arguments that a system is adequately dependable in a given application and environment [12], [13]. Assurance cases are widely required by regulation for safety-critical systems in the EU. There have been several graphical notation systems proposed for assurance cases. GSN (Goal Structuring Notation) and CAE (Claim, Argument, Evidence) are such two notation systems, and a standardization effort for these notation systems have been attempted in OMG (Object Management Group). Attempts to give formal semantics to this notation have been published recently [14].

When elaborating an assurance case for any safety-related system containing software, five principles must be followed [12]. The first four principles can be considered in isolation, while the fifth principle (called Principle 4+1 by the authors) is transversal to the others.

1) Software safety requirements shall be defined to address the software contribution to system hazards. [Requirement validity]
2) The intent of the software safety requirements shall be maintained throughout requirements decomposition. [Requirement decomposition]
3) Software safety requirements shall be satisfied. [Requirements satisfaction]
4) Hazardous behaviour of the software shall be identified and mitigated. [Hazardous software behaviour]
5) The confidence established in addressing the software safety principles shall be commensurate to the contribution of the software to system risk. It is necessary to provide evidence that the previous four principles have been followed. [Confidence]

Note that the first three principles concern traditional and important aspects of requirements engineering, such as elicitation, decomposition and verification of requirements. The fifth principle is also related to RE through the notions of traceability and accountability, which are needed in order to collect evidence supporting the safety case.

B. Sustainability

Sustainability has emerged as an area of significant concern regarding the potential consequences for humanity as a result of the rapid depletion of the planet Earth's finite natural resources, coupled with exponential economic and population growth [15]. The concept of sustainability has also emerged as an area of research within the field of computing as a result of the pervasiveness and dependency of software systems in society [16]. For example, the development of sustainable software has been identified as one of the key challenges in the field of computational science and engineering [17]. While the importance of sustainability is increasingly recognized, many software systems are unsustainable, and the broader impacts of most software systems on sustainability are unknown. However, the concept of sustainability is a term that remains ambiguous and widely abused sixteen years after the Brundtland Commission [15].

Derived from the Latin sustinere, sustainability can be defined as 'capable of being endured' and 'capable of being 'maintained' [18]. This simplicity of this definition fails to capture the complexity of the the concept that can be viewed from a range of different dimensions [19]:

- **Economic**: relates to financial aspects and business value. It includes capital growth and liquidity, questions of investment, and financial operations..
- **Environmental**: relates to the use of and care for natural resources. It includes questions ranging from immediate waste production and energy consumption to the balance of local ecosystems and concerns of climate change.
- **Individual**: relates to individual freedom and agency (the ability to act in an environment), human dignity and fulfillment. It includes the ability of individuals to thrive, exercise their rights and develop freely.
- **Social**: concerns relationships between individuals and groups. For example, this aspect covers the structures of mutual trust and communication in a social system and the balance between conflicting interests.
- **Technical**: relates to the ability to maintain and evolve artificial systems (such as software) over time. This refers to maintenance and evolution, resilience, and the ease of system transitions.

While there is currently no agreed definition of the concept of sustainability within the field of computing there is growing consensus that sustainability should be considered as a first-class, non-functional requirement [20] [1]. While this position has been suggested by a number of commentators [21] [22], it has been made without explicit reference to the characteristics or qualities that sustainability would be composed of. In contrast, Venters et. al., [23] defined software sustainability as a composite, non-functional requirement which is a measure of a systems:

- **Extensibility**: the software's ability to be extended and the level of effort required to implement the extension;
- **Interoperability**: the effort required to couple software systems together.
- **Maintainability**: the effort required to locate and fix an error in operational software;
- **Portability**: the effort required to port software from one hardware platform or software environment to another;
- **Reusability**: the extent to which software can be reused in other applications;
- **Scalability**: the extent to which software can accommodate horizontal or vertical growth.
- **Usability**: the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use.

However, one of the principal challenges in defining sustainability as a non-functional requirement is how to demonstrate that the quality factors have been addressed in a quantifiable way.

The discussion of how to define sustainability is not limited to the field of computing [24]. It is suggested that to understand sustainability, we need to ask which system to sustain, for whom, over which time frame, and at what cost [25]. The point of departure for defining sustainability is the second law of thermodynamics, which states that the state of entropy of the entire universe, as an isolated system, will always increase over time [26]. Sverdrup and Svensson [27] argue that in developing rules and criteria for sustainability, it is important to shape these as a set of principles which are free of value judgments and cultural biases. Becker et. al., [6] argue the software profession lacks a common ground that articulates its role in sustainability design. To address this they proposed a set of sustainability principles to contribute to the conversation on the role of the software systems in undermining and in enabling a sustainable future for our planet.

III. Related Work

While there is a substantial body of work focusing on either safety or sustainability domain, up to now only a few pieces of research have considered the parallels between safety and sustainability disciplines. These are discussed below.

One of the first studies to report such parallels is Mahaux et al. [28], which was an exploratory study on how well some RE techniques work for representing sustainability. The study showed that *misuse case sheets* (a well known hazard analysis technique) can be used for revealing sustainability requirements. Similarly, Hessami et al. [29] proposes an integrated perspective on sustainability through a systems framework. In that framework, the authors consider that a system is sustainable in a low-level sense if it can ensure "system security and safety to resist or tolerate deteriorating or disruptive internal or external threat conditions", among other characteristics. So, in both works, safety is perceived as one of the aspects of sustainability.

Van Gorp [3] looks at the safety and sustainability ethical issues that engineers deal with during the design process. According to the author, both issues are related to utility and general rights and that engineers should learn about "what the affected actors value relative to the product that is being designed" and try "to take care of these valued things".

Other works, such as Frakes and Kang [30] and Banerjee et al. [31] mention safety and sustainability as important characteristics of systems, but make no comparison between these concepts. Additionally, they refer to sustainability as a means to maintain a technical system on a long-term basis, which is an indicator that they restrict their understanding of sustainability to its technical dimension. Banerjee et al. go a bit further, stating that long term maintenance should be achieved while using green sources of energy.

Finally, the closest related work is Penzenstadler et al. [1], which compares the history and concepts of sustainability with those of security and safety. They argue that while sustainability is not a completely emergent property, as safety, it has to deal with emergent aspects manifested as second-and third-order effects. Both also require an in-depth exploration of the problem space and application domain to find solutions. In addition, they share a few myths, such as translating into quality goals that compete with other system goals or that they can be treated in isolation from other qualities. Finally, the authors argue that sustainability requires quality assurance techniques comparable to those for safety, and that it should be included in software engineering standards, as is safety.

The present paper discusses the relationship between safety and sustainability in greater depth than the available literature and explores how a wider range of methods and tools for safety can be used or adapted to sustainability.

IV. Parallels Between Safety and Sustainability

As discussed above, safety and sustainability are complex, multifaceted domains. In order to explore the parallels between them, we structure our discussion according to four different dimensions: systemicity, complexity, certification and social vs. business perception. The similarities, differences and synergies observed are summarized in Table I.

A. Systemic Property

A system is often defined as set of interacting or interdependent components forming an integrated whole and relationships with a common purpose [32]. As per this definition, a system is:

- made up of components: parts that make up a system i.e. subsystems;
- components are interrelated: one part of the system depends on its' one or more other parts;
- has a boundary, which clearly defines the internal and external limits of the system;
- has a purpose, which is the overall function of the system; operates in an environment, which is external to the system and is influenced by or influences the system;
- has interfaces, these are point of contact where a system meets its environment or where subsystems meet each other;
- operates under constraints, these are the limits to what a system can accomplish within its environment.

A substantial amount of research has been completed in software systems engineering on software modularity and interaction, which essentially aims to realise complex systems through simpler interacting parts in such a way that the interdependencies between the parts are minimised (so called low coupling principle) and the closely related functions and properties are collected into one part (so called high cohesion principle). Yet, SE is also well familiar with the set of properties which do not fit into any single part of a system, but relate to the system as a whole. Such properties are said to be systemic: for a system to provide for this property, each of its sub-parts must do so. Examples of such properties are response time, security, and, as discussed below, safety, and sustainability.

	Similarities	Differences	Synergies
Systemic property	Both are emergent properties. Full satisfaction of both properties is unfeasible.	They have different orders of effect, which manifest in different timescales. Definition of the system scope and the identification of stakeholders is more complex for sustainability.	Sociotechnical approaches to safety.
System complexity	High interactive complexity causes lack of understanding of event chains leading to failure/unsustainability. Difficult decomposition of both safety and sustainability goals into software requirements.	Safety is more susceptible to single fault problems. Leverage points are better understood in safety thanks to the notion of severity in risk analysis.	Development of techniques for decomposition of safety/sustainability (inspired by safety assurance principles). Joint research on GSN and other goal-based formalisms for sustainability.
Certifications & standards	Increasing importance of regulations in both fields	Safety standards are more mature and widespread. Lack of widely accepted quantifiable indicators of sustainability. Sustainability standards lack support for stakeholders identification in IT systems.	Generalization of the safety assurance principles, if applicable to sustainability assurance. Adoption/adaptation of existing reliability indicators.
Social& business perception vs. Costs	Responsibility is moving towards organizations, not the individual.	Safety is better understood and promoted, and the extra cost of safety is tolerated. Some aspects of sustainability can never be directly observable.	There is a need to exemplify sustainability in IT systems and their effects: examples and cases studies proved crucial in the promotion of safety.

Table I: Summary of the analysis

1) Safety: In safety-related domains, safety is treated not as a property of a single component or of a certain part of a given system, but one that the system has to fulfill as a whole. For instance, hazards are identified and defined over end-to-end functions, and only afterwards are decomposed into software safety goals. Although safety is considered an emergent property, in reality what can be observed externally is the *lack* of safety, which manifests as a violation of the stated safety objectives.

Since a system cannot be inherently safe, the goal of safety is to guarantee that the probability of an accident is as low as reasonably practical [7]. Through proper analysis, design and development, one can ensure only that all known risks have been removed with a certain budget and up to a certain acceptable level. To ensure that the development process and design are “proper”, each application domain (e.g., aerospace, medical, or chemical product production) that considers safety a pertinent property, provides a detailed process and product regulation/legislation [13].

In order to analyze safety issues, each safety engineering process will work within clearly defined system boundaries. These boundaries are defined during requirements engineering by an analysis of the surrounding system context and its resource flows and functions. Studies on safety are normally concerned with the immediate effects of the system (due to its direct functions/properties, also called 1st order effects), and those effects that are enabled due to the system (also called indirect or 2nd order effects) [33]. The cumulative safety effects that could occur due to prolonged and mass use of the system are rarely considered, barring cases where dangerous materials are involved, or alike.

Safety engineering does consider the environment/context within which a given system has to operate. Indeed, it is not possible to reason fully about safety or the lack of it without knowing certain operational details such as: who is going to use the system (user profile), how the system is going to be used (potential operation), for how long it is going to be used (mission time), what type of training the user has, etc. All this information is elicited and captured during requirements engineering.

2) Sustainability: Sustainability too, is systemic, in that for a system to be sustainable all of its sub-parts need to address sustainability as well. Yet, the scope of the system that concerns sustainability is much harder to define; this is because ultimately every socio-technical system consumes certain natural resources, engages societies, and, inevitably, produces waste. Since one of the sustainability dimension is concerned with the environment, all resource consumption and waste generation is pertinent to the sustainability of the planet as a whole. While the consideration of a planetary scope is clearly not feasible for each software systems engineering project, ignoring it could also lead to a false sense of achievement. For instance, exploring electronic waste, or moving data centers to another country does not improve the sustainability of the phone companies, though their effects may not be felt directly by the communities using them in their services. Thus, to help set the scope of sustainability, one must determine what must be sustained, for whom, for how long, and at what cost? [25]. Yet, while these questions are helpful for knowing what to address, providing meaningful answers to them is still a challenge.

Similar to safety, sustainability cannot be assured without setting the context. Questions normally asked during requirements engineering, like *who will be using the system, in which environment, and what are the cultural/organisational norms within that environment?*, are very important for the analysis of social sustainability perspectives; whether or not renewable resources are used and/or non-renewables wasted/polluted, etc.

As noted in Section 2, the concept of sustainability comprises 5 dimensions of the system. Interestingly, though these viewpoints are inherently linked, a system can be sustainable in one (or more) of them, and at the same time be unsustainable in others. The best example for this is the economic dimension: a socio-technical system (such as petroleum extraction companies) can be extremely profitable for its owners and shareholders delivering exceptional economic sustainability yet might cause dreadful environmental effects. Similarly, exceptional technical sustainability of software assets can be achieved at the expense of the economic dimension.

Similar to safety, a system cannot be proven to be sustainable. This is because the changing context of the system will inevitably affect the sustainability of the system itself. For instance, changing technology (like new programming languages or environments) or changing user requirements will degrade the technical sustainability of software assets, as well as (eventually) their economic sustainability (if their maintenance costs too much). Thus, sustainability can only be assessed for the given time and state of the system. Such assessments must be repeated periodically throughout system use, expecting that eventually (in time) the un-sustainability will be observed [34]. In other words, changes in requirements must also be assessed with respect to sustainability.

B. System complexity

Giving a general definition of complexity has proven to be difficult [35]. In the context of system design, *complexity* is typically defined in terms of the number of elements that constitute the system and the interaction among them and with the environment. According to [36], a complex system is a system that fulfills several of these conditions:

- A high number of components;
- strong interaction among the component;
- long operation time;
- diversity and variability of the components;
- demanding environment;
- multiplicity of activities/objectives.

Because of these features, it is common for complex systems to exhibit emergent behaviors, i.e. behaviors that can be observed only once all the components are interconnected and the system is operated in a certain context. This is a consequence of the difficulty of reasoning about the cause-effect chains within systems of these characteristics.

1) Safety: According to Leveson, there are many factors that are increasing the complexity of current safety-critical systems. Many of them are caused by the widespread use of software [11].

- The fast pace of technological change is increasing the diversity and variability of the system components, while reducing the time for training and realizing both the potential and the risks of these new technologies.
- Changing nature of accidents. Safety originated in the field of process engineering, where system hazards are typically derived from well-understood and observable physical laws. In contrast, digital systems introduce new failure modes which are much more unpredictable in nature. Overconfidence in redundancy and misunderstanding of failure models of software-implemented components have been related to recent aerospace accidents [11].
- New types of hazards have appeared, for instance related to loss or corruption of information, which go beyond the traditional conception of hazard as uncontrolled or undesired release of energy.
- Decreasing tolerance for single accidents. As technology becomes more predominant in our daily life, it also becomes possible to potentially harm increasing numbers of people and impact future generations. Nuclear and chemical plants are good examples of very demanding safety-critical systems.
- Increasing interactive complexity and high coupling. The potential interactions among the components of a computer system very often cannot be thoroughly understood or anticipated. Sometimes complicated systems need to be built, which actually exceed the human intellectual ability to reason. Sometimes it is the information about the system that is incomplete, for instance due to the use of legacy code or because of partial knowledge of the operational environment. The tight coupling of software components allows disruptions or dysfunctional interactions in one part of the system to impact distant parts of the system, making hazard analysis more complicated.
- The new, and more sophisticated, relationships between humans and automation make it more difficult to reason about potential interaction problems, creating new human errors and a new distribution of human errors. Humans are still in charge of most of the decision-making processes, while computers are responsible for the automated implementation of those decisions. Adequate communication and HMI are required in order to ensure system observability and prevent accidents due to a false sense of safety.

Most techniques for safety analysis and safety design help in handling complexity. Hazard analysis and risk analysis are techniques that are used during requirements engineering for identifying the critical system functions, thus allowing the system designer to disregard those functions that are not relevant from a safety perspective. Fault-tree analysis and Failure Mode and Effects Analysis help in identifying the elements of the system that are relevant for a certain hazard, which helps in reducing the design and the verification efforts. The purpose of modeling techniques such as GSN is to provide a way to visualize and analyze the complex relationships between arguments of a safety assurance case.

However, it has been argued that as complexity increases, techniques based on event chains, like the ones mentioned above, may be too simple. A new accident model based on systems thinking (STAMP) has been defined by Leveson [5].

This new model relies on control theory notions and integrates sociotechnical aspects.

2) Sustainability: As a systemic property, sustainability is inherently complex. Its complexity comes from the need to consider the system and its context from at least five perspectives, as well as its three orders of effect on these perspectives [33].

As in safety, sustainability requires the cooperation of all elements involved. That means not only hardware and software components, but also the people interacting with that software system and the processes surrounding it.

A parallel can also be drawn with respect to the decomposition of the concern. In requirements engineering, sustainability goals may be defined at different levels of abstraction [37]. High-level concerns (e.g., contribute to a healthy environment) may be decomposed into lower level goals (e.g., optimize energy consumption) and realized by different design strategies (e.g., by using Energy Star qualified hardware [38], by defining a process that reduces physical waste, or by writing efficient algorithms).

Furthermore, making sure that all elements fulfill their sustainability goals also requires a clear assignment of responsibilities: from the roles responsible for the software development to the users of that system. Eventually, verifying the achievement of these goals requires traceability of both their decomposition and responsibility assignment.

Finally, as with any complex issue, a number of strategies will have to be combined to achieve both safety and sustainability. Small, isolated efforts towards sustainability can be jeopardized by greater sustainability risks or by safety hazards, when these are not taken into account during the elaboration of the system requirements. For example, writing efficient algorithms might be irrelevant when the system encourages mass consumption of unnecessary good. Yet, there are subtle differences. While, in safety, a small loophole, such as a floating point conversion, can expose the system to great unsafety, in sustainability this is less likely to happen. In other words, failing to provide sustainability will, in general, not manifest as an accident or mishap; especially not immediately. Punctual efforts, such as the recycling of obsolete hardware, contribute to sustainability, but have little leverage when compared to challenging the purpose of the system, for example. Requirements engineers are in a great position to point at potential impact and ask questions that might encourage the stakeholders to consider sustainability.

C. Certifications & standards

The goal of *certification* is, in general, to confirm that the characteristics of some object, person or organization conform to a certain definition, or *standard*. Certification is typically performed externally, by governmental certification agencies. Software, standards usually define the desirable properties of a software product or, alternatively, of a software development process. Adherence to these standards is mandatory in certain fields, such as transportation, energy or healthcare, because of their high social impact.

1) Safety: Safety is a strongly regulated field, with multiple national and international safety standards applied at different levels. So-called functional safety standards concern the elimination of hazards caused by Electrical, Electronic and Programmable Systems. Some authors have discussed and compared existing functional safety standards at length [8], [39].

This large amount of standards and regulations shows that responsibility for safety is shifting from the individual to government. Individuals no longer have the ability (or knowledge) to control the risks around them and are demanding that government assume greater responsibility for controlling technology. The application of these standards generates tension with industrial goals, like short time to market or reduced cost.

Each functional safety standard typically defines its own process for system certification, but according to [13], they can all assimilated to the five safety assurance principles discussed in Section 2: requirements validity, requirements decomposition, requirements satisfaction, hazardous software behaviour and confidence.

2) Sustainability: Standards and certifications on sustainability play a similar role to standards in safety: adherence to them may be needed to convince customers, authorities and the society of commitment to sustainability.

A number of reasons may drive organizations to seek compliance with sustainability standards. On particular, companies might value a sustainable corporate image, in response to a growing demand by the society for organizations with social and environmental corporate practices [40].

There are a number of standards related to sustainability. Rodrigues and Penzenstadler (2013) observe that their focus vary from sustainability development [41], [42], to sustainability reporting [43], [44], to sustainable design [45], [46], among others. However, most of them are specific to particular industry sectors [47] and sustainability dimensions, such as the ISO 14000 for Environmental Management [41] or the ISO 26000 for Social Responsibility [42]. The authors also observe that software systems and their supporting infrastructures are often not explicitly accounted for in the life-cycle analysis (LCA) of sustainable projects. Instead, LCA normally uses tangible items in their analysis.

One difference worth pointing out is that safety standards can be powerful requirements engineering tools, as they typically help with stakeholder identification and selection of metrics for the development of safety-critical IT systems, e.g. mean time between failures. However, in sustainability, standards are either: (1) concerned with a business organization and its operations [41], [42], or (2) or devoted to reducing the use of hazardous material, ensuring energy efficiency, and reducing waste in electronic products [48], [38], [49]. We are missing standards that support the analysis and design of sustainable IT systems, beyond of the selection of green hardware.

D. Social & Business Perception vs. Costs

Both safe and sustainable systems share, in a broad sense, a common objective: not to harm the environment. The social perception of risk plays an important role when it comes to decide what can be spent in order to fulfill this goal.

1) Perceptions and Costs of Safety: From a technological perspective, safety design adds on costs, since additional work is required, not only for design and verification of the safety goals elicited during requirements engineering, but also importantly for safety assurance. As stated in the fifth principle (4+1) of Section 2, providing confidence about the definition and verification of safety requirements is needed for safety assurance. The ability to collect this information is a significant difference with respect to other software industry. Businesses would reduce this cost when possible, but since the adherence to standards provides access to markets, it can as well be considered an investment.

Socially, it is acceptable to pay for a safer product; for example, an automobile with additional safety features is normally more expensive than one without them, so consumers are prepared to take on the extra cost that business will pass on to them. In terms of marketing, the reputational (social) damage of unsafe goods is very large, so safety costs are also investment into avoiding these losses.

In the SE community, the professional recognition of safety is very high, and it is intimately related to the notion of software bug. Examples of incidents and accidents caused by software bugs are described in lectures and notebooks, and students become familiar with them. Classical examples from many different domains, like the Path Finder and Ariane 5 from aerospace, the Patriot missile from military, Therac-25 from medicine, or the Toyota throttle problem from automotive are part of graduate education and successfully convey the importance of generating high-quality code for critical applications.

In companies developing safety-critical products, safety awareness is very high, it permeates the whole development process and affects all the people involved. This includes customer service, management, organization and engineering activities and is shared by software providers as well.

2) Perceptions and Costs of Sustainability : Similar to the safety domain, high profile sustainability failure cases have driven regulation and business costs. For instance, practices that caused visible water and soil pollution have been tightly regulated in most countries. In such cases, where the impact of unsustainability is apparent, business has had to accept cost of sustainability maintenance.

However, due to the indirectness and longer time scale taken for manifestation of enabling and systemics effects, much of the cost of unsustainable business conduct remains hidden and so, is unaccounted for. For instance, the cost of deforestation in the Amazon forest, where locals clear the forest to make space for crop fields, is distributed across time and space, arguably resulting in accelerated climate change, but is not immediately observed by the logging communities. Similarly, inequality in access to education or health-care leads to loss of potential contribution from talented but disadvantaged individuals, reducing the prosperity of a nation, but this is not directly observed by each member of that nation and does not disturb those who gain from this inequality.

Nevertheless, in recent years, the true costs of sustainability effects have increasingly been gaining recognition. As a results, consumers have become more willing to accept higher costs and express preferences for sustainable products and services (such as organic food, recycled paper, renewable energy, etc.). The political and government bodies have started to impose longer-term sustainability targets and regulations on waste management and use of natural resources.

Sustainability now features as a prominent topic in the recent research funding calls (e.g. Horizon 2020 Work Programme (2014 – 2015), European Commission FP7 Work Programme (2013) for ICT). Sustainability in the software context is also a fast growing research area. A recent systematic mapping study has shown an increase on the number of publications on sustainability and software engineering: from 82 papers published in the last 25 years, 70 belong to the period of 2010 to 2013 [1]. One of the main challenges to be addressed for Software Engineering, is to establish a clear cause-effect traceability between software engineering activities/processes and the sustainability effect of the produced software. Such relationship is well exemplified in safety, where connection between safety engineering and accident prevention is clearly demonstrated. Requirements engineering has a prominent role in this challenge, as it shapes the scope and the purpose of the system, and therefore, can have the greatest leverage in supporting sustainability.

V. Conclusions

This paper provided an explorative analysis of similarities, differences, and potential synergies between safety and sustainability. That way, software engineers, and in particular requirements engineers, can start taking lessons from safety engineering and apply them to sustainability design for software systems.

We reviewed the most important concepts of functional safety, hazard analysis and safety engineering, and the principles of safety assurance. Then, we discussed parallels between safety and sustainability, namely the characteristic as systemic property, their respective complexity, the issue of certification and standards, and the social and business perception versus costs.

The results of our comparisons are summarized in Table I. Although still far from being exhaustive, this analysis represents a step forward in order to improve the tools and methods available for software engineers addressing sustainability issues.

As future work, we envision to apply safety engineering techniques to example systems and evaluate their applicability in a sustainability context. We also wish to investigate how the concepts of safety assurance can be exploited and adapted to sustainability, considering even the convenience of having sustainability cases, inspired by the notion of safety case. A comparison between GSN and other goal oriented techniques for RE will also be carried out. Finally, through this study we have learned about Resilience Engineering as an extension of safety towards systems thinking [50]. This innovative approach to safety (and beyond) shows promising synergistic potential with sustainability and will be explored further.

VI. Acknowledgments

This work is partially supported by the European Social Fund; Ministry of Science, Research and the Arts Baden-Württemberg; FP7 ASCETiC project (No. 610874); FAPERJ and CNPQ funding agencies; Swedish Knowledge Foundation (KKS), within project SaDIES.

References

[1] B. Penzenstadler, A. Raturi, D. Richardson, and B. Tomlinson, "Safety, security, now sustainability: The nonfunctional requirement for the 21st century," *Software, IEEE*, vol. 31, no. 3, pp. 40–47, May 2014.

[2] N. G. Leveson, "Software safety: Why, what, and how," *ACM Computing Surveys (CSUR)*, vol. 18, no. 2, pp. 125–163, 1986.

[3] A. V. Gorp, *Ethical issues in engineering design; safety and sustainability*. Springer, 2005.

[4] P. M. d'Orey and M. Ferreira, "Its for sustainable mobility: A survey on applications and impact assessment tools," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 15, no. 2, pp. 477–493, 2014.

[5] N. Leveson, *Engineering a safer world: Systems thinking applied to safety*. Mit Press, 2011.

[6] C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, B. Penzenstadler, N. Seyff, and C. C. Venters, "Sustainability design and software: The karlskrona manifesto," in *ICSE'15: Proceedings of the International Conference on Software Engineering*, 2015.

[7] N. G. Leveson, *Safeware: system safety and computers*. ACM, 1995.

[8] D. Smith and K. Simpson, *Functional safety*. Routledge, 2004.

[9] J. Rushby, "Critical system properties: Survey and taxonomy," *Reliability Engineering & System Safety*, vol. 43, no. 2, pp. 189–219, 1994.

[10] J. H. Saleh, K. B. Marais, and F. M. Favaró, "System safety principles: A multidisciplinary engineering perspective," *Journal of Loss Prevention in the Process Industries*, vol. 29, pp. 283–294, 2014.

[11] N. Leveson, "A new accident model for engineering safer systems," *Safety science*, vol. 42, no. 4, pp. 237–270, 2004.

[12] R. Hawkins, I. Habli, and T. Kelly, "Principled construction of software safety cases," in *SAFECOMP 2013-Workshop SASSUR (Next Generation of System Assurance Approaches for Safety-Critical Systems) of the 32nd International Conference on Computer Safety, Reliability and Security*, 2013, p. NA.

[13] R. Hawkins, I. Habli, T. Kelly, and J. McDermid, "Assurance cases and prescriptive software safety certification: A comparative study," *Safety science*, vol. 59, pp. 55–71, 2013.

[14] Y. Matsuno, "A design and implementation of an assurance case language," in *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, June 2014, pp. 630–641.

[15] D. Meadows, J. Randers, and D. Meadows, *Limits to Growth: The 30-Year Update*. Earthscan, 2005.

[16] N. Chue-Hong, "We are the 92%," in *The Second Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE2)*, New Orleans, LA USA, November 2014, keynote. [Online]. Available: http://figshare.com/articles/We_are_the_92_/1243288/

[17] WSSSPE, "Workshop on sustainable software for science: Practice and experiences," http://wssspe.researchcomputing.org.uk/. [Online]. Available: http://wssspe.researchcomputing.org.uk/

[18] O. Dictionaries, *Oxford English Dictionary*. Oxford University Press, 2011.

[19] C. Becker, S. Betz, R. Chitchyan, L. Duboc, S. M. Easterbrook, B. Penzenstadler, N. Seyff, C. C. Venters, and S. A. Kocak, "Requirements: The key to sustainability," 2015, unpublished.

[20] N. Amsel, Z. Ibrahim, A. Malik, and B. Tomlinson, "Toward sustainable software engineering," in *ICSE: Proceedings of the 33rd International Conference on Software Engineering*, Waikiki, Honolulu, HI, USA, 2011, pp. 976–979.

[21] S. Naumann, M. Dick, E. Kern, and T. Johann, "The greensoft model: A reference model for green and sustainable software and its engineering," *Sustainable Computing: Informatics and Systems*, vol. 1, no. 4, p. 294–304, December 2011.

[22] C. Calero, M. F. Bertoa, and M. Ángeles Moraga, "Sustainability and quality: Icing on the cake," in *RE4SuSy: Proceedings of the Second International Workshop on RE for Sustainable Systems*, Rio de Janeiro, Brazil, 2013.

[23] C. C. Venters, L. M. S. Lau, M. Griffiths, V. Holmes, R. Ward, C. Jay, C. Dibsdale, and J. Xu, "The blind men and the elephant: Towards an empirical evaluation framework for software sustainability," *Journal of Open Research Software*, vol. 2, pp. 1–6, 2014.

[24] C. C. Venters, C. Jay, L. M. S. Lau, M. K. Griffiths, V. Holmes, R. R. Ward, J. Austin, C. E. Dibsdale, and J. Xu, "Software sustainability: The modern tower of babel," in *RE4SuSy: Proceedings of the Third Int. Workshop on RE for Sustainable Systems*. Karlskrona, Sweden: CEUR-WS 1216, 2014.

[25] J. A. Tainter, "Social complexity and sustainability," *ecological complexity*, vol. 3, no. 2, pp. 91–103, 2006.

[26] A. Eddington, *Space, Time and Gravitation: An Outline of the General Relativity Theory*. Butler Press, 2007.

[27] H. Sverdrup and M. G. E. Svensson, "Defining the concept of sustainability - a matter of systems thinking and applied systems analysis," in *Systems Approaches and Their Application*, M. O. Olsson and G. Sjöstedt, Eds. Springer, 2005, pp. 143–164.

[28] M. Mahaux, P. Heymans, and G. Saval, "Discovering sustainability requirements: an experience report," in *Requirements engineering: foundation for software quality*. Springer, 2011, pp. 19–33.

[29] A. Hessami, F. Hsu, and H. Jahankhani, "A systems framework for sustainability," in *Global Security, Safety, and Sustainability*, ser. Communications in Computer and Information Science, H. Jahankhani, A. Hessami, and F. Hsu, Eds. Springer Berlin Heidelberg, 2009, vol. 45, pp. 76–94. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04062-7_9

[30] W. Frakes and K. Kang, "Software reuse research: status and future," *Software Engineering, IEEE Transactions on*, vol. 31, no. 7, pp. 529–536, July 2005.

[31] A. Banerjee, K. Venkatasubramanian, T. Mukherjee, and S. Gupta, "Ensuring safety, security, and sustainability of mission-critical cyber-physical systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 283–299, Jan 2012.

[32] A. Waring, *Practical Systems Thinking*. Thomson, 1996.

[33] L. M. Hilty and B. Aebischer, "Ict for sustainability: An emerging research field," in *ICT Innovations for Sustainability*. Springer, 2015, pp. 3–36.

[34] B. Penzenstadler, "Software Engineering for Sustainability," Habilitation Thesis, Technische Universität München, 2015.

[35] Y. Bar-Yam, *Dynamics of complex systems*. Perseus Books, 1997.

[36] J. Ladyman, J. Lambert, and K. Wiesner, "What is a complex system?" *European Journal for Philosophy of Science*, vol. 3, no. 1, pp. 33–67, 2013.

[37] B. Penzenstadler and H. Femmer, "A Generic Model for Sustainability with Process- and Product-specific Instances," in *First Intl. Workshop on Green In Software Engineering and Green By Software Engineering*, 2013.

[38] "EnergyStar," http://www.energystar.gov/about, accessed: 2015-06-08.

[39] P. Baufreton, J. Blanquart, J. Boulanger, H. Delseny, J. Derrien, J. Gassino, G. Ladier, E. Ledinot, M. Leeman, P. Quéré *et al.*, "Multi-domain comparison of safety standards," in *Proceedings of the 5th International Conference on Embedded Real Time Software and Systems (ERTS2), Toulouse, France*, 2010.

[40] K. L. Becker-Olsen, "The csr conundrum: understanding consumer response to corporate social responsibility," in *Handbook of Research on Marketing and Corporate Social Responsibility*, R. P. Hill and R. Langan, Eds. Edward Elgar Publishing Limited, 2014, vol. 1, pp. 149–174.

[41] "ISO 14000 - Environmental management," 2004, http://www.iso.org/iso/home/standards/management-standards/iso14000.htm.

[42] "ISO 26000 Guidance on social responsibility," 2010, http://www.iso.org/iso/home/standards/iso26000.htm.

[43] *Sustainability Reporting Guidelines*, Version 3.1 ed., Global Reporting Initiative, Amsterdam, Netherlands, 2006.

[44] C. Herzig and S. Schaltegger, "Corporate sustainability reporting. an overview," in *Sustainability Accounting and Reporting*, S. Schaltegger,

M. Bennett, and R. Burritt, Eds. Springer Netherlands, 2006, vol. 21, pp. 301–324. [Online]. Available: http://dx.doi.org/10.1007/978-1-4020-4974-3_13

[45] U. G. B. Council, "Leadership in energy and environmental design (leed)," http://www.usgbc.org/.

[46] U. G. S. Administration, "Sustainable design," www.gsa.gov/sustainabledesign.

[47] FTSE Group, "The industry classification benchmark (icb)," 2012, http://www.icbenchmark.com/.

[48] "RoHS Guide," http://www.rohsguide.com/, accessed: 2015-06-08.

[49] "EPAT verification," http://www.epeat.net/resources/verification/, accessed: 2015-06-08.

[50] E. Hollnagel, D. D. Woods, and N. Leveson, *Resilience engineering: Concepts and precepts*. Ashgate Publishing, Ltd., 2007.