Safety, Security, Now Sustainability: The Nonfunctional Requirement for the 21st Century

Birgit Penzenstadler, Ankita Raturi, Debra Richardson, and Bill Tomlinson, University of California, Irvine

// Software engineers can considerably improve civilization's sustainability by taking into account not just the first-order impacts of software systems but also their second- and third-order impacts. //

IN THE LAST decades of the 20th century, software became deeply ingrained in a wide range of human activities. The growing ubiquity of complex software systems led to both software security and software safety failures, which in due course led to the need for software engineering researchers and practitioners to address both security and safety during the development of large-scale software systems.[1,2] In particular, security and safety were added to the set of nonfunctional requirements, whereas taxonomies of nonfunctional requirements at that time focused on other qualities such as efficiency, reliability, and usability. For example, IEEE Standard 730-1989 stated that the development of "critical software—that is, where failure could impact safety or cause large financial or social losses" should be supported, but it didn't explicitly address safety or security as qualities.[3] ISO/IEC 9126-1:2001 on software quality listed security but not safety.[4] Finally, ISO/IEC 25010:2011 included safety as an explicit characteristic.[5]

In the 21st century, software systems are central to the operation of most sectors of industrial society. As such, they're embedded in and enable the rather unsustainable way in which most of us live. It isn't civilization's intention to harm the Earth, but the collective sum of our individual actions, which often favor local convenience over global responsibility, added to the effects of the societal structures we've created, lead to negative consequences for our environment. If our civilization is to transition to sustainability, many sectors of society will need to rethink their modes of operation. The ubiquity of software offers a unique opportunity: a shared point of intervention across a wide range of intertwined and impactful industries. To support the transition to sustainability, environmental sustainability must be explicitly considered as a nonfunctional requirement in the software engineering process.

Software engineering has considerable potential to support "greening through IT"—that is, making civilizations more environmentally sustainable via IT interventions.[6] To draw attention to such issues in software engineering, we argue that sustainability must be treated as a first-class quality alongside other critical attributes such as safety, security, efficiency, reliability, and usability.

Sociotechnical IT systems are among the most powerful tools humanity has ever created. Understanding how to engineer the requirements of such systems so that they better enable social well-being and sustainability could have a constructive benefit to the world. Instead of merely optimizing current systems, software engineers must embrace *transition engineering*—an emerging discipline that enables change from existing unsustainable systems to more sustainable ones by adapting and filtering demand to a declining supply.[7]

Why Sustainability Matters

The United Nations Earth Summit of 1992 led to the establishment of the UN Framework Conventions on Climate Change and Biological Diversity, which 195 countries ratified but few applied. The UN Rio+20 Conference took place 20 years later and resulted in recommendations that prescribed an objective: to preserve our world for future generations, we must change consumer behavior with regard to environmental impact, where action is currently limited compared to other economic activities. These recommendations, however, were limited in their practical use.

Psychologist Geoffrey Beattie's answer to the question, "Why we aren't saving the planet yet?" is that the problem is too complex for a single mind to solve, and therefore we shy away from becoming active.[8] Sustainably engineered software systems could help facilitate a more sustainable lifestyle by directly influencing the surrounding system context (including the people, infrastructure, and environment). Furthermore, software systems have such a significant impact on our everyday lives that changes toward more environmental sustainability can ripple to other systems with which they interact and also affect other industries. This impact can be direct, be indirect, or occur as a rebound effect. This potential for positive change is why software engineers should care about supporting sustainability (see the "A Characterization of Sustainability" sidebar).

Sustainability science is currently discussing whether sustainability is limited by ecological constraints or enabled by continuous innovation and transformation.[9] Joseph Burger and his colleagues explain the ecological constraints as a flow of resources from the environment to society that must conform to physical laws and balance.[10] This is reflected as information flows in software systems whenever we represent real-world objects with natural resources—for example, in a production site. John H. Matthews and Frederick Boltz reflect the optimistic view that we can sufficiently adapt human societies to overcome resource limitations by innovatively transforming our systems.[11] As software engineers, we must accept the challenge of integrating sustainability into the systems we build. The ubiquity of software systems in industrial civilization means that the reach of this effort could be quite broad indeed.

Lorenz Hilty and his colleagues' "Sustainability and ICT—An Overview of the Field" presents both ecological pessimism and technological optimism when it differentiates environmental informatics, green (in and by) IT/ICT, and sustainable human–computer interaction.[12] Their bottom-line analysis is that technological efficiency alone won't produce sustainability; rather, efficiency in combination with sufficiency will support innovation for sustainability.

How can we tackle the goal of effectively supporting sustainability through software development? It will take a systematic approach that ranges from incorporating sustainability requirements to performing quality assurance to ensure that these requirements are adequately realized, and all of this should be guided by an explicit sustainability quality characteristic. Susan Krumdieck stated that similar arguments were brought up for security and safety.[7] We claim that the same applies to sustainability. Consequently, the question we address is how might we learn, adapt, and build on past research regarding security and safety requirements to establish sustainability requirements and thereby make our world more sustainable through software engineering for sustainability?

History of Safety and Security

Safety and security have systematically been included in software engineering due to serious accidents and other negative effects of software systems:

- Software engineers started to consider safety issues in their processes, particularly hazard and fault analysis, following a series of failures in which users were hurt or systems were damaged.[2,13]
- Software engineers began working on security algorithms and mechanisms after security threats led to widespread identity theft and financial fraud, among other issues.[1]

A CHARACTERIZATION OF SUSTAINABILITY

In general, sustainability is the capacity to endure, but interpreting this requires context. A popular definition of sustainable development was given by the UN as "meeting the needs of the present without compromising the ability of future generations to meet their own needs."[1] This definition was characterized by economic, social, and environmental dimensions. Robert Goodland extended this characterization by including the human dimension, which portrays individual development by all humans throughout their lives.[2] When analyzing IT system sustainability, these four dimensions apply along with an additional technical dimension that supports better structuring of concerns with respect to software systems. More elaborate theoretical frameworks for defining sustainability exist, all of which rely on systems thinking to some extent.[3–9] The main text of this article focuses on environmental sustainability because other aspects of sustainability are, in part or implicitly, already supported by established software engineering practices.

Considering sustainability with regard to software engineering is two-fold: although improving the sustainability of the software development process will have some impact when compared to the overall impact that software systems affect in our daily life, improving software systems' sustainability in their surrounding context will allow software engineering to have a more significant impact on improving the sustainability of human society.

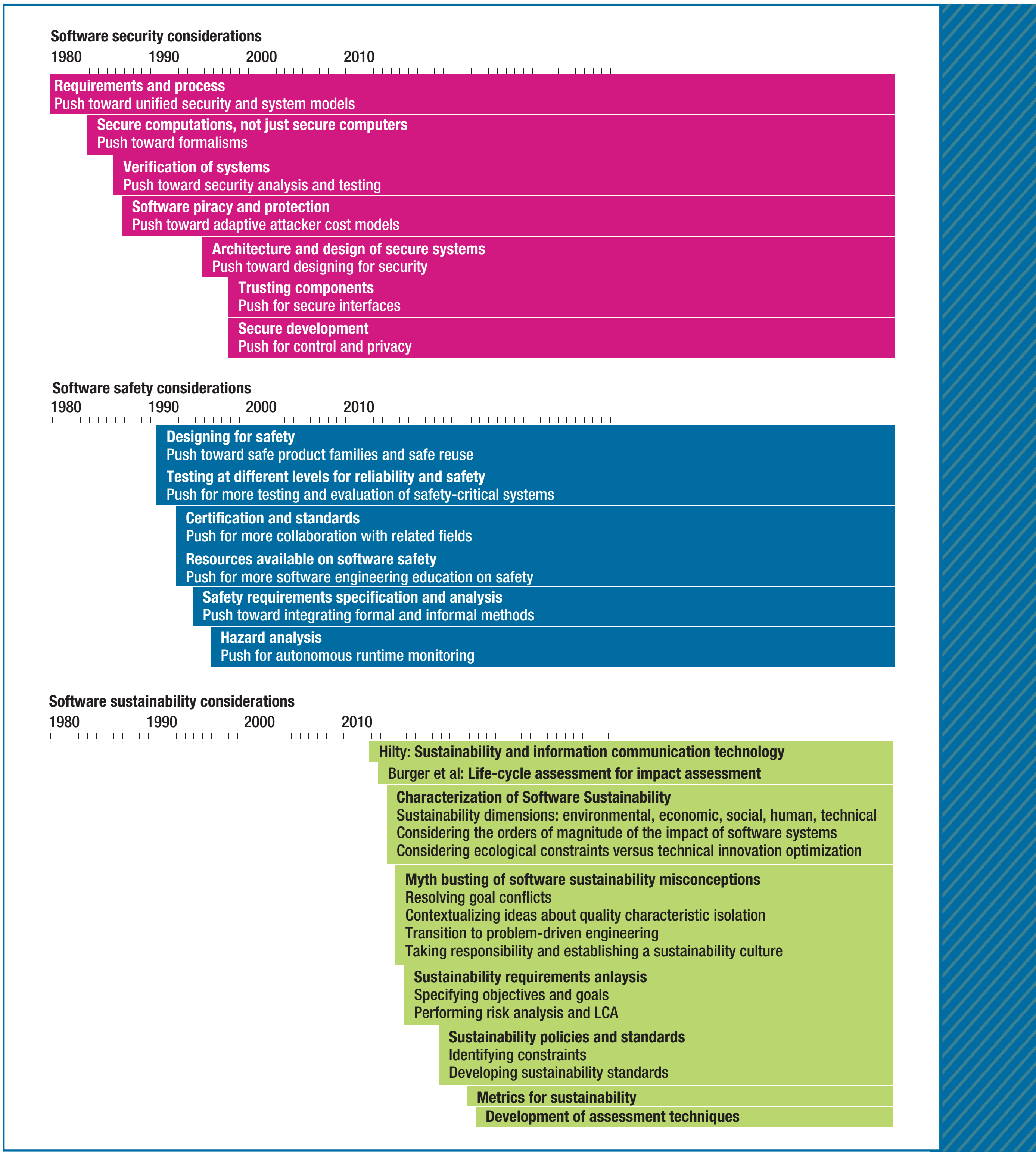
With regard to a software system's impact on sustainability, we distinguish three orders of magnitude.[10] First-order impacts are direct effects of a software system on its environment—for example energy usage, e-waste production, emissions caused by required infrastructure, and so on. Second-order impacts are the indirect effects or induction effects that software systems cause—for example, changes to user resource consumption or consumer behavior. Third-order impacts are rebound effects—for example, the increased efficiency of systems that tend to make us use even more systems which, in total, consume even more energy. All these must be considered within an encompassing approach to supporting sustainability. A variety of standards are in place or under development to help design these means of support.[11–14]

References

1. "Our Common Future," report, United Nations, 1987; www.un-documents.net/our-common-future.pdf.
2. R. Goodland, "Sustainability: Human, Social, Economic and Environmental," *Encyclopedia of Global Environmental Change: Social and Economic Dimensions,* vol. 5, T. Munn, ed., Wiley, 2002, pp. 489–491.
3. P. Burger and M. Christen, "Towards a Capability Approach of Sustainability," *J. Cleaner Production*, vol. 19, 2001, pp. 787–795.
4. M. Christen and S. Schmidt, "A Formal Framework for Conceptions of Sustainability—A Theoretical Contribution to the Discourse in Sustainable Development," *Sustainable Development*, vol. 20, no. 6, pp. 400–410.
5. A. Dobson, "Environment Sustainabilities: An Analysis and a Typology," *Environmental Politics*, vol. 5, no. 3, 1996, pp. 401–428.
6. K.-H. Robert et al., "Strategic Sustainable Development—Selection, Design and Synergies of Applied Tools," *J. Cleaner Production*, vol. 10, no. 3, 2002, pp. 197–214.
7. S. Bell and S. Morse, *Sustainability Indicators: Measuring the Immeasurable?*, Earthscan, 1999.
8. D. Mebratu, "Sustainability and Sustainable Development: Historical and Conceptual Review," *Environmental Impact Assessment Rev.*, vol. 18, no. 6, 1998, pp. 493–520.
9. D.H. Meadows, *Thinking in Systems—A Primer*, Earthscan, 2008.
10. L.M. Hilty et al., "The Relevance of Information and Communication Technologies for Environmental Sustainability," *Environmental Modelling and Software*, vol. 21, no. 11, 2006, pp. 1618–1629.
11. *ISO 14000, Environmental Management*, ISO, 2004.
12. *ISO 26000:2010, Guidance on Social Responsibility*, ISO, 2010.
13. *IEEE Std. 1680-2009, Environmental Assessment of Electronic Products,* IEEE, 2009.
14. *ISO/IEC 25010:2011, Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models*, ISO, 2011.

In both cases, it took a while before these characteristics were included in requirements engineering standards. In IEEE Standard 830, safety and security are only mentioned as types of constraints in all three editions (1984, 1994, and 1998).[14] The most recent requirements engineering standard, ISO/IEC/IEEE 29148:2011, addresses both security and safety as central topics.[15] The graphical timeline in Figure 1 illustrates the progression of the most important phases in safety and security development with respect to requirements and process.[1,2]

Safety. A *safety requirement* is a constraint derived from identified hazards. Nearly all the serious accidents that software has been involved with over the past 20 years can be traced to requirements flaws, not coding errors.[13] The reason is that the requirements usually reflect incomplete or wrong assumptions about the operations or context. Extensive investigation into specification and analysis of requirements for safety-critical systems began in the 1990s, especially in the area of formal methods.[2]



Software security considerations
1980
1990
2000
2010
Requirements and process
Push toward unified security and system models
Secure computations, not just secure computers
Push toward formalisms
Verification of systems
Push toward security analysis and testing
Software piracy and protection
Push toward adaptive attacker cost models
Architecture and design of secure systems
Push toward designing for security
Trusting components
Push for secure interfaces
Secure development
Push for control and privacy
Software safety considerations
1980
1990
2000
2010
Designing for safety
Push toward safe product families and safe reuse
Testing at different levels for reliability and safety
Push for more testing and evaluation of safety-critical systems
Certification and standards
Push for more collaboration with related fields
Resources available on software safety
Push for more software engineering education on safety
Safety requirements specification and analysis
Push toward integrating formal and informal methods
Hazard analysis
Push for autonomous runtime monitoring
Software sustainability considerations
1980
1990
2000
2010
Hilty: Sustainability and information communication technology
Burger et al: Life-cycle assessment for impact assessment
Characterization of Software Sustainability
Sustainability dimensions: environmental, economic, social, human, technical
Considering the orders of magnitude of the impact of software systems
Considering ecological constraints versus technical innovation optimization
Myth busting of software sustainability misconceptions
Resolving goal conflicts
Contextualizing ideas about quality characteristic isolation
Transition to problem-driven engineering
Taking responsibility and establishing a sustainability culture
Sustainability requirements anlaysis
Specifying objectives and goals
Performing risk analysis and LCA
Sustainability policies and standards
Identifying constraints
Developing sustainability standards
Metrics for sustainability
Development of assessment techniques

FIGURE 1. The history of safety, security, and sustainability considerations in software. Software security has been addressed from the 1980s onward, beginning with concerns about secure computation and verification. Software safety has been considered since the early 1990s from a design perspective. Sustainability (as characterized in the sidebar) has just recently started to receive recognition as a quality objective for software systems.

Security. A *security requirement* is a "manifestation of a high-level organizational policy into the detailed requirements of a specific system."[1] Although security models have been around since the 1970s, elaboration on security requirements has been added only recently as an afterthought and often as a compromise to satisfy compliance to standards. Integrating security requirements with systems engineering helps ensure that a model of domain ontology is constructed first to drive the rest of the requirements process.[1]

Implications. First, we realize that some aspects of safety and security can be handled by more traditional quality attributes. For example, although the maxim "keep spurious alarms to a minimum"[13] is safety-related, it can also be labeled as a performance requirement.

Second, not every requirements engineering process requires safety or security, but it's generally accepted that these quality characteristics should be explicit in addition to the traditional characteristics, as Boehm's and McCall's quality models and ISO 25010 illustrate (see the sidebar).[5]

Nevertheless, safety and security are called out and treated specifically because they're important characteristics. We argue that the same is true for sustainability, especially the dimension of environmental sustainability, and that we need to find adequate means to analyze, support, verify, and validate sustainability requirements in software engineering.

Comparing Sustainability to Safety and Security

Certain safety and security concepts can also be applied to sustainability, including

- characterizations,
- strategies to overcome myths and misconceptions,
- methods for requirements analysis and quality assurance, and
- policies and standards that impose constraints but also support goal realization.

Figure 1 presents an overview of past and present considerations of safety, security, and sustainability.

Characterizations

Safety is "an emergent property that arises when the system components interact within an environment."[13] An *emergent property* is a characteristic that's only exhibited by the system in use, as opposed to an inherent characteristic already exhibited in the system's static existence. In other words, a software system isn't unsafe in itself, but it can be unsafe in its application environment.[13]

In contrast, sustainability isn't completely an emergent property but must be dealt with in two parts: an inherent part that shows up in the first-order and some second-order effects, and the emergent part that manifests itself in second- and third-order effects (see the sidebar for an explanation of the three orders of magnitude).

Security is the degree of resistance to, or protection from, harm to information (for example, unauthorized access). A security policy is the set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information. A similar mechanism of policies and requirements is needed for supporting and enforcing environmental sustainability.

An additional challenge in characterizing sustainability arises from the fact that measuring first-order effects is easier than measuring second-order effects (as they occur only in combination with human behavior) or, even more difficult, third-order effects, which require measuring systemic impacts.

Myth Busting

Myths and misconceptions that apply to safety and security can also apply to sustainability in the areas of goal conflicts, isolation, solution-driven engineering, and culture.

Goal conflict. Nancy Leveson wrote, "A classic myth is that safety conflicts with achieving other goals and that tradeoffs are necessary to prevent losses. In fact, this belief is totally wrong. Safety is a prerequisite for achieving most organizational goals, including profits and continued existence. ... The 'conflict' myth arises because of a misunderstanding about how safety is achieved and the long-term consequences of operating under conditions of high risk."[13]

Similarly, there's a misperception that sustainability will conflict with a company's economic goals. However, a practical view of how we deal with environmental impacts that companies cause shows that such conflicts only occur because companies don't have to pay for the emissions they produce during manufacturing, nor are they responsible for the disposing of products at the end of their life cycles. If companies had to include these expenses, sustainability would form a natural part of their economic bottom line.

Isolation. A second common misperception is that we can deal with safety, security, and sustainability as separate from other quality characteristics. In isolation, requirements are replaced by solution-specific constraints—in other words, we, as software engineers, set the scope too small so that not all related aspects are taken into account, resulting in a suboptimal solution. For example, we often mistreat security requirements by replacing them with security-specific architectural constraints that can hinder the security team from using the most appropriate security mechanism to deal with actual security requirements.[16] Security and safety engineers by now are aware that they have to look at the "real" requirements instead of simply replacing them with solution elements; the same applies to sustainability requirements.

Solution-driven engineering. Security requirements are often solution mechanisms in disguise[16]—"The user logs into the system with his security code" isn't a good security requirement. It leads to a suboptimal situation because a solution is already being prescribed rather than the problem space being analyzed with separate solution finding. The same holds for sustainability requirements—for example, we might apply energy-saving mechanisms instead of questioning and analyzing the process that requires the mechanism. It's necessary to explore in depth both the problem space and the application domain to find optimal solutions.

The culture of blame and denial. Establishing a responsible safety culture prevents the tendency to blame others after safety incidents occur.[13] A similar process can also be applied to security breaches. In contrast, hardly anyone accepts responsibility for environmental sustainability or damage, so we're talking about sustainability, culture, and responsibility. We must develop a culture of responsibility for our planet that includes everyone where the still-present and strongly counterproductive denial of sustainability issues ceases to exist because the arguments for sustainability and the indicators of various collapsing systems are already visible.

Institutional change can't be solved within requirements engineering, but it might be a trigger for systemic improvements.

Requirements Analysis

If sustainability is a quality attribute, then we need analysis and quality assurance techniques comparable to what we have for safety and security. To analyze sustainability requirements, we can make use of parallels in defining objectives and goals as well as risk analysis techniques.

Objectives and goals. In safety, system-theoretic process analysis uses intent specifications that contain the safety-related goals and objectives for the system under consideration.[13]

For sustainability goals and objectives, we've developed a generic reference model that allows engineers to structure sustainability goals according to the five dimensions described in the sidebar (economic, social, environmental, human, and technical) and to detail them further with requirements and constraints or supporting activities (see www.se4s.org for an overview).[17,18] Our model allows engineers to link the environmental sustainability goals more strongly to other sustainability goals. Furthermore, as with safety goal specifications, the systematic decomposition and derivation of requirements allows for traceability and validation of objectives during quality assurance.

Risk analysis and life-cycle assessment. In safety, three hazard-analysis techniques are widely used for risk analysis: fault-tree analysis, event-tree analysis, and hazards and operability analysis.

A common technique that sustainability science uses is life-cycle assessment (LCA), which is based on systems thinking.[10] The insights from LCA might be beneficial in developing sustainable software systems because LCA helps identify risks and hazards, which provides the basis for defining sustainability requirements and constraints.

Various researchers use misuse cases for security requirements elicitation.[19,20] They can use this technique in addition to LCA to identify sustainability risks that might otherwise be overlooked during the analysis. They must deal with these identified risks by specifying constraints that avoid threatening conditions.

Policies and Standards

Specifying constraints is a common practice to incorporate regulations from policies and standards into safety and security requirements specifications. Constraints assure that these regulations are adhered to during quality assurance. They're derived from identified hazards as a way to avoid accidents. A method to deal with risks and hazards in sustainability management would also be to derive constraints from regulatory sources.

Aiming for environmental sustainability standards to support constraint specification, the next step is to extend software engineering standards. For example, ISO/IEC/IEEE 29148:2011 on requirements engineering could explicitly include a section on sustainability in the software requirements specification document template.[15]

To ensure that such regulations and standards have the desired effects, it might be necessary to trigger institutional change so that environmental regulations consider more than first-order effects. Institutional change can't be solved within requirements engineering, but it might be the appropriate trigger for systemic improvements.

Outlook

We've described a number of areas in software engineering that can borrow from safety and security to address sustainability in effective ways; each of these areas requires extended further research. In particular, this article has explored sustainability as a nonfunctional requirement, along with consequent requirements analysis and standards. Quality assurance techniques corresponding to the requirements are also needed, which necessitates future research in establishing sustainability metrics as well as assessment techniques.

Thorough research requires evaluating various methods through case studies in a variety of application domains to develop guidance on the most appropriate, adapted methods in the context of supporting environmental sustainability as it pertains to software systems.

Metrics

For both standards and quality assurance, we need to define a set of metrics for the different dimensions of sustainability by relying on the respective sets of metrics available, including the

- ISO 14000 family for environmental sustainability,[21]
- ISO 26000 for social sustainability,[22] and
- Environmental Sustainability Index (http://sedac.ciesin.columbia.edu/data/collection/esi).

The IEEE 1680 family of standards for environmental assessment of IT is already taking steps in that direction.[23] Furthermore, there's a need for standards that give guidance for development, as is done through IEC 61508 on the functional safety of electronic safety-related systems.[24]

Assessment Techniques

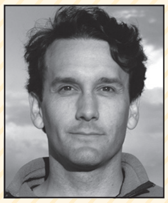
For assessment techniques to address quality assurance, we propose evaluating and adapting LCA to software engineering and making use of environmental impact assessment in software engineering. LCA can be

ABOUT THE AUTHORS

BIRGIT PENZENSTADLER is a postdoctoral researcher at the University of California, Irvine (UCI). Her research interests include supporting sustainability from within requirements engineering and quality assurance. Penzenstadler received a PhD in informatics from Technische Universitat München. She's a member of IEEE. Contact her at bpenzens@uci.edu.

ANKITA RATURI is a PhD student in software engineering at UCI. Her research interests include involving sustainability considerations in software development, software standards, and metrics. Raturi received an MS in informatics from UCI. Contact her at araturi@uci.edu.

DEBRA RICHARDSON is a professor of informatics at UCI. Her research interests include exploring how software engineering can be made to address socially relevant problems such as sustainability. Richardson received a PhD in computer and information science from the University of Massachusetts, Amherst. Contact her at djr@uci.edu.



BILL TOMLINSON is a professor of informatics at UCI. His research interests include environmental informatics, human-computer interaction, multiagent systems, and computer-supported learning. Tomlinson received a PhD in media arts and sciences from MIT. Contact him at wmt@uci.edu.

applied to software systems in two scopes: it can either be applied to the development process of exclusively the software itself, without consideration of the process it supports, which would lead to an assessment of the development environment (developer tools, coding standards, quality assurance). Or LCA can be applied to the process in the operational environment that the software system supports, which would put a strong environmental focus on business process analysis.

Sustainability in software engineering not only encompasses energy efficiency and green IT but must also consider the second- and third-order impacts of software systems. To do so, sustainability must be considered as a first-class quality attribute and specified as a nonfunctional requirement of IT systems.

Considering the five dimensions of sustainability—human, social, environmental, economic, and technical—all but one can be supported to some extent by current software engineering methods because the concerns are not new, per se. Missing from consideration is environmental sustainability. While few people deliberately intend to harm the environment, it's often more convenient to take the easy way out by failing to explore environmentally friendlier options, given that we already have so many advanced technologies and facilities that science and the economy have brought us. This seems to be true in current software engineering methods as well.

The challenge of incorporating this conflict into software engineering is an issue of requirements prioritization, which is usually handled by negotiation between system stakeholders. Perhaps the solution requires an explicit stakeholder for environmental sustainability. Another way to ensure prioritization of environmental sustainability is to enforce policies based on the compliance-driven economy. Although we can hope that all citizens take responsibility for their impact on the planet, we know it will be a long time before this comes to fruition.

If sustainability policies and standards are put in place and software engineers prioritize them in the systems they develop, future technology may significantly contribute indirectly to influencing the behavior of users who interact with those systems in some ways while directly contributing to saving the planet.

Acknowledgments

This work is part of the DFG EnviroSiSE project (grant number PE2044/1-1) and was funded in part by the National Science foundation grant no. 0644415.

References

1. P.T. Devanbu and S. Stubblebine, "Software Engineering for Security: A Roadmap," *Proc. Future of Software Eng.*, 2000, pp. 227–239.
2. R. Lutz, "Software Engineering for Safety: A Roadmap," *Proc. Conf. Future of Software Eng.* (ICSE 00), 2000, pp. 213–226.
3. *IEEE Std. 730-1989, Software Quality Assurance Plans*, IEEE, 1989.
4. *ISO/IEC 9126-1:2001, Software Engineering—Product Quality—Part 1: Quality Model*, ISO, 2001.
5. *ISO/IEC 25010:2011, Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation—System and Software Quality Models*, ISO, 2011.
6. B. Tomlinson, *Greening through IT*, MIT Press, 2010.
7. S. Krumdieck, "The Survival Spectrum: The Key to Transition Engineering of Complex Systems," *Proc. ASME Int'l Mechanical Eng. Congress and Exposition*, 2011.
8. G. Beattie, *Why Aren't We Saving the Planet?: A Psychologist's Perspective*, Routledge, 2010.
9. G. Mace, "The Limits to Sustainability Science: Ecological Constraints or Endless Innovation?" *PLOS Biology*, vol. 10, no. 6, 2012, e1001343.
10. J. Burger et al, "The Macroecology of Sustainability," *PLoS Biol*, vol. 10, no. 6, 2012, pp. 2–7.
11. J.H. Matthews and F. Boltz, "The Shifting Boundaries of Sustainability Science: Are We Doomed Yet?" *PLOS Biology*, vol. 10, no. 6, 2012, e1001344.
12. L. Hilty, W. Lohmann, and E. Huang, "Sustainability and ICT—An Overview of the Field," *Proc. 25th Ann. EnviroInfo Conf.*, 2011, pp. 13–28.
13. N. Leveson, *Engineering a Safer World: Systems Thinking Applied to Safety*, MIT, 2011.
14. *IEEE 830-1998, Recommended Practice for Software Requirements Specifications*, IEEE, 1998.
15. *ISO/IEC/IEEE 29148:2011, Systems and Software Engineering—Life Cycle Processes—Requirements Engineering*, ISO, 2011.
16. D. Firesmith, "Engineering Security Requirements," *J. Object Technology*, vol. 2, no. 1, 2003, pp. 53–68.
17. B. Penzenstadler and H. Femmer, "A Generic Model for Sustainability with Process- and Product-Specific Instances," *Proc. 2013 Intl. Workshop Green in/by Software Eng.* (GIBSE 13), 2013, pp. 3–8.
18. B. Penzenstadler and H. Femmer, "RE@21: Time to Sustain!" *Proc. 2nd Int'l. Workshop Requirements Eng. for Sustainable Systems*, 2013, pp. 4–12.
19. N.R. Mead and T. Stehney, "Security Quality Requirements Engineering (Square) Methodology," *Proc. 2005 Workshop Software Eng. for Secure Systems–Building Trustworthy Applications* (SESS 05), 2005, pp. 1–7.
20. G. Sindre and A. Opdahl, "Eliciting Security Requirements with Misuse Cases," *Requirements Eng.*, vol. 10, no. 1, 2005, pp. 34–44.
21. *ISO 14000, Environmental Management*, ISO, 2004.
22. *ISO 26000:2010, Guidance on Social Responsibility*, ISO, 2010.
23. *IEEE Std. 1680, Environmental Assessment of Electronic Products*, IEEE, 2013.
24. *IEC 61508, Functional Safety*, IEC, 2010.

See www.computer.org/software-multimedia for multimedia content related to this article.