# Requirements: The Key to Sustainability

**Christoph Becker**, University of Toronto

**Stefanie Betz**, Karlsruhe Institute of Technology

**Ruzanna Chitchyan**, University of Leicester

**Leticia Duboc**, State University of Rio de Janeiro

**Steve M. Easterbrook**, University of Toronto

**Birgit Penzenstadler**, California State University, Long Beach

**Norbert Seyff**, University of Applied Sciences and Arts Northwestern Switzerland

**Colin C. Venters**, University of Huddersfield

*// Software's critical role in society demands a paradigm shift in the software engineering mind-set. This shift is driven by requirements engineering. //*

**SOFTWARE SYSTEMS** are a major driver of social and economic activity. Software engineering (SE) tends to focus on the technical elements—artificial systems with clear boundaries and identifiable parts and connections, modules and dependencies. But software systems are embedded in other technical systems and in socioeconomic and natural systems. This embedding is obvious when the interaction is explicit, such as environmental monitoring or flight control software.

However, software-intensive systems have become so essential to societies that the resulting sociotechnical systems' boundaries and interactions are often hard to identify. For example, communication, travel booking, and procurement systems influence the socioeconomic and natural environment through far-reaching effects on how we form relationships, how we travel, and what we buy. The engineering process rarely makes these effects explicit. Their lack of visibility makes assessing a software system's long-term and cumulative impacts difficult.

Designing for sustainability is a major challenge that can profoundly change SE's role in society. But what does it mean to establish sustainability as a major concern in SE? As software engineers, we're responsible for our software's long-term consequences, irrespective of the primary purpose of the system we're designing. Requirements are the key leverage point for practitioners who want to develop sustainable software-intensive systems. Here, we present two examples that illustrate the changes needed in SE and show how considering sustainability explicitly will affect requirements activities.

## Sustainability in Software Engineering

Sustainability is the capacity to endure, so a system's sustainability describes how well it will continue to exist and function, even as circumstances change. Sustainability has often been equated with environmental issues, but it's increasingly clear that it requires simultaneous consideration of environmental resources, societal and individual well-being, economic prosperity, and the long-term viability of technical infrastructure.

A technical system's sustainability differs considerably from that of a socioeconomic system. Software


0740-7459/16/$33.00 © 2016 IEEE

engineers tend to focus on sustainability's technical dimension, in which it's simply a measure of the software system's longevity.[1] However, to understand broader sustainability issues, we must ask which system to sustain, for whom, over which time frame, and at what cost.[2] This involves five interrelated dimensions:[3]

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

Complex software-intensive systems can affect sustainability in any of these dimensions. Changes in one system, in one dimension, often have impacts in other dimensions and other systems. For example, consider a hard-to-maintain software system (technical sustainability). Excessive maintenance costs affect the owning company's financial liquidity (social and economic sustainability). This might limit its growth and even threaten its survival (economic sustainability).

Similar tradeoffs occur across other dimensions. For example, carbon offsets incentivize environmentally sustainable behavior through tradeoffs with the economic dimension. The triple-bottom-line perspective requires a business to account for social and environmental as well as financial outcomes.[4] The corresponding business practices have led to a surge in the number of social enterprises, which achieve survival rates above average for new businesses.[5]

> We need to consider systems' immediate features and effects and their longer-running aggregate and cumulative impact.

Increasingly, software engineers need to understand the effects by which software system design decisions can enable or undermine the sustainability of socioeconomic and natural systems over time (see the sidebar, "Classifying the Systemic Effects of Software"). Because sustainability is inherently multidisciplinary, any effort to define it involves concepts, principles, and methods from a range of disciplines and makes an integrated view crucial for effective system design. The notion of sustainability design brings these concerns together using systems-thinking principles (see the sidebar, "Sustainability Principles for Software Engineering").

## A Tale of Two Projects

A software system's impact on its environment is often determined by how the software engineers understand its requirements. This impact's foundation is set in the decisions on which system to build (if any at all), the choices of whom to ask and whom to involve, and the specification of what constitutes success.

The following examples describe two projects to develop a procurement system that supports purchasing products and contracting services in a private company in the energy sector. Products, services, and suppliers must pass the company's approval process and be registered in the system before a purchase. This approval considers the supplier's reliability, capacity to deliver, and, in some cases, adherence to international standards of environmental management, health, and safety.

The examples are inspired by a real-world project.[6] The first example reflects typical software projects, which don't use sustainability design. The second shows what could happen if a project applied sustainability design. Terms in italics indicate aspects that are common to both projects, for easy comparison.

### Development without Sustainability Design

The *project's purpose* is to maximize the organization's procurement efficiency, increase the financial return, and ensure suppliers' compliance

## CLASSIFYING THE SYSTEMIC EFFECTS OF SOFTWARE

Many critical effects in sociotechnical systems play out over time. So, we need to consider not just our systems' immediate features and effects but their longer-running aggregate and cumulative impact. We distinguish three orders of effects.[1]

*Immediate effects* are the direct effects of the production, use, and disposal of software systems. This includes the immediate benefit of system features and the full life-cycle impacts, such as a life-cycle assessment (LCA) would include. An LCA evaluates the environmental impact of a product's life from the extraction of raw materials to its disposal or recycling.

*Enabling effects* arise from a system's application over time. This includes not only opportunities to consume more (or fewer) resources but also other changes induced by system use.

*Structural effects* represent "persistent changes observable at the macro level. Structures emerge from the entirety of actions at the micro level and, in turn, influence these actions."[1] Ongoing use of a new software system can lead to shifts in capital accumulation; drive changes in social norms, policies, and laws; and alter our relationship with the natural world.

Consider Airbnb.com. Its immediate effects include resources consumed and jobs created during its development, energy consumed during its deployment, and the room renting and booking services it offers. Its enabling effects include changes in how its users make travel arrangements as alternatives to hotel bookings and in how property owners rent out space.

These enabling effects (the "sharing economy") have been both praised and criticized for their far-reaching structural impacts. For example, Airbnb represents a substantial share of the buy-to-let market in major cities. The continuing price surges in these cities' hot spots have been linked to the density of buy-to-let properties. Many of these exist only because of the arbitrage that services such as Airbnb.com provide. The system enables transactions that provide a higher return on investment than long-term rentals. This has caused major concerns in several large cities.

### Reference

1. L.M. Hilty and B. Aebischer, "ICT for Sustainability: An Emerging Research Field," *ICT Innovations for Sustainability*, Springer, 2015, pp. 3–36.

with certain rules. The criteria for selecting products and services focus on price, delivery time, and payment conditions.

Using a *stakeholder* influence matrix, the project leader focuses on those stakeholders who can "stop the show." A few influential stakeholders determine the project *scope* early on so that the project can focus on a minimal design scope to maximize project speed. The project team moves swiftly to determine the *boundaries* of the software to be; the only scoping questions revolve around the software's interfaces with neighboring systems.

The project's *success criteria* are to develop and deliver the system within the given budget and time. The question of feasibility centers on the software project investment's expected amortization period. *Risk analysis* focuses on economic risks that could inhibit project completion.

*Requirements elicitation* requests stakeholders' input through structured forms to identify what they want the system to do. Additionally, the team analyzes previous systems and consults business process documents. Requirements prioritization is determined by functional requirements and economic constraints and is completed quickly because the core stakeholder group has a strong consensus.

The *requirements specification* is documented following the software requirements specifications template from IEEE Standard 830. System *measurement and monitoring* employ performance and availability indicators. The system is completed on time and within budget and shows a reasonably low rate of faults, so the project is considered a success at completion.

### Development with Sustainability Design

Consider conducting the same project while treating sustainability as a first-class concern in line with sustainability design principles (see the sidebar, "Sustainability Principles for Software Engineering").

While discussing the *project's purpose*, the initial project team discusses the company's values and responsibilities and identifies opportunities to support the

company's sustainable development. For example, the system can support sustainability in the supply chain by making transparent the carbon footprint of purchases and facilitating the selection of providers who apply sustainable practices. This doesn't change the overall project objectives, but it influences subsequent steps.

The *scope* of analysis starts with an inclusive, integrated view of the procurement processes, material flows into the company, and the local community's social and political environment. When defining possible system *boundaries*, the team experiments with multiple perspectives and works jointly with the procurement department and others.

The team expands the set of *stakeholders* and draws on knowledge beyond the team by using a stakeholder impact analysis. This analysis considers enabling and structural effects to identify those most affected by the project, including those external to the company. Stakeholders include local supplier representatives, service delivery organizations, process analysts, the chief technology officer, and the strategic-planning and foresight group.

To keep the number of stakeholders manageable, a sustainability expert acts as a surrogate stakeholder for others in the community and the further environment that the system might affect. A team member is assigned to each of the five sustainability dimensions so that responsibility for identifying possible effects is clear and effective communication with additional stakeholders can take place. These team members consult relevant experts in areas such as supply chain sustainability, carbon accounting, and socially responsible procurement. They also consult anthropologists analyzing and interpreting current technological developments and their impact on society.

The team agrees that the project's *success criteria* are not restricted to whether it's delivered on time and within budget, but will be *measured and monitored* over the 36 months after project completion. In this period, the team will measure a set of indicators covering the five sustainability dimensions. It will try to measure

## SUSTAINABILITY PRINCIPLES FOR SOFTWARE ENGINEERING

The following principles are based on "Sustainability Design and Software: The Karlskrona Manifesto."[1]

- Sustainability is systemic; a system can never be treated in isolation from its environment.
- Sustainability is multidimensional; the five key dimensions are economic, social, environmental, technical, and individual.
- Sustainability is interdisciplinary; sustainability design in software engineering requires an appreciation of concepts from other disciplines and must work across disciplines.
- Sustainability transcends the software's purpose; any software can impact the sustainability of its socioeconomic, sociotechnical, cultural, and natural environments.
- Sustainability is multilevel; it requires us to consider at least two spheres during system design: the system under design and its sustainability, and the wider system of which it will be part.
- Sustainability is multi-opportunity; it requires us to seek interventions that have the most leverage on a system[2] and to consider the opportunity costs.
- Sustainability involves multiple timescales; it requires long-term thinking to address the timescales on which sustainability effects occur.
- Sustainability isn't zero-sum; changing a system's design to consider the long-term effects doesn't automatically imply making sacrifices now.
- System visibility is a necessary precondition and enabler for sustainability design. This is because only a transparent status of the system and its context, made visible at different abstraction levels and perspectives, can enable system designers to make informed responsible choices.

For more on this, see www.sustainabilitydesign.org.

### References


1. C. Becker et al., "Sustainability Design and Software: The Karlskrona Manifesto," *Proc. 37th IEEE Int'l Conf. Software Eng.* (ICSE 15), 2015, pp. 467–476.
2. D.H. Meadows, *Leverage Points: Places to Intervene in a System*, Sustainability Inst., 1999.

- technical debt,
- social reputation and improved relations with the local community,
- individual aspects such as privacy compliance and the satisfaction of those involved in the procurement process,
- environmental aspects such as the total carbon footprint of the products and services acquired, and
- amortization of the project costs and improved cost–benefit relations in procurement.

During *risk analysis*, the team considers internal and external risks related to systemic effects in all five dimensions. For example, considering the evolving regulations on environmental accountability as a risk, the team develops a set of transparency requirements for the system. It also identifies uncertainties about future shifts in procurement as sustainable products become more competitive. So, it includes a feature to monitor these uncertainties.

During *requirements elicitation*, the team employs participatory techniques. The inclusive perspective lets the project leverage contributions from a broader set of stakeholders, including local service providers. In a series of workshops, the team uses a sustainability reference goal model to derive specific sustainability goals for the project and align them with other system goals, while deriving extended usage scenarios with the local community representatives.

The resulting *requirements specification* is based on a template that includes checklists for sustainability criteria and standards compliance in all five dimensions. The document is circulated among all the stakeholders and is shared with regulatory agencies to demonstrate that the project meets relevant sustainability rules. So, it's also used more actively in subsequent stages.

## Sustainability Debt

The system resulting from this procurement project is different when development takes into account sustainability principles and therefore long-term consequences.

Focusing on sustainability design, software engineers must adopt a mind-set quite different from the puzzle-solving attitude often found in engineering and business. Now, the objective is to identify and understand "wicked problems": problems that are deeply embedded in a complex system with no definitive formulation and no clear stopping rule. In such cases, every solution changes the nature of the problem, so little opportunity exists for trial-and-error learning.[7,8] Instead, we need an adaptive, responsive, and iterative approach emphasizing shared understanding.

Figure 1 highlights selected immediate, enabling, and structural effects of the procurement system in the five sustainability dimensions. Consider a system feature that tracks individual products' carbon footprint, letting users choose products with lower footprints. The compound structural effect in the economic dimension can benefit local suppliers with environmentally sustainable production and can lead to a reduced carbon footprint.

The diagram in Figure 1 supports interactive collaboration among stakeholders to discover, document, and validate the system's potential effects. Not all effects will be positive. For example, automating product selection rules to minimize the carbon footprint takes away the manager's freedom to make decisions in the procurement process.[9] This can reduce mutual trust between the organization's members.

The diagram also facilitates a conversation about *sustainability debt*: decisions made for the present situation have invisible effects that accumulate over time in each of the five dimensions.[10] When we increase energy consumption, reduce individual privacy, impose technical barriers, or incur additional financial costs, we incur debts in these dimensions to different stakeholders. Making these effects visible is the first step to understanding and considering them in system design decisions.

## Requirements Are the Key

In those two projects, a series of decision points occurred during system design. Many of them were requirements-engineering activities that occurred repeatedly in all iterations throughout the projects. Each decision influenced the decision space of subsequent choices and profoundly affected the system and its effects. Table 1 highlights how key activities change when we consider sustainability design principles.

Requirements' leverage becomes clear when we consider their relationships with engineering techniques. We develop techniques to quantify, construct, and test artifacts and to control whether the results fall in an acceptable range. However, for design concerns such as usability, performance, maintainability, or sustainability, such techniques are only applied once a need has been identified. Without such a need, the engineering techniques will remain unused and hence have no effect on the project.

For example, techniques for increasing technical sustainability

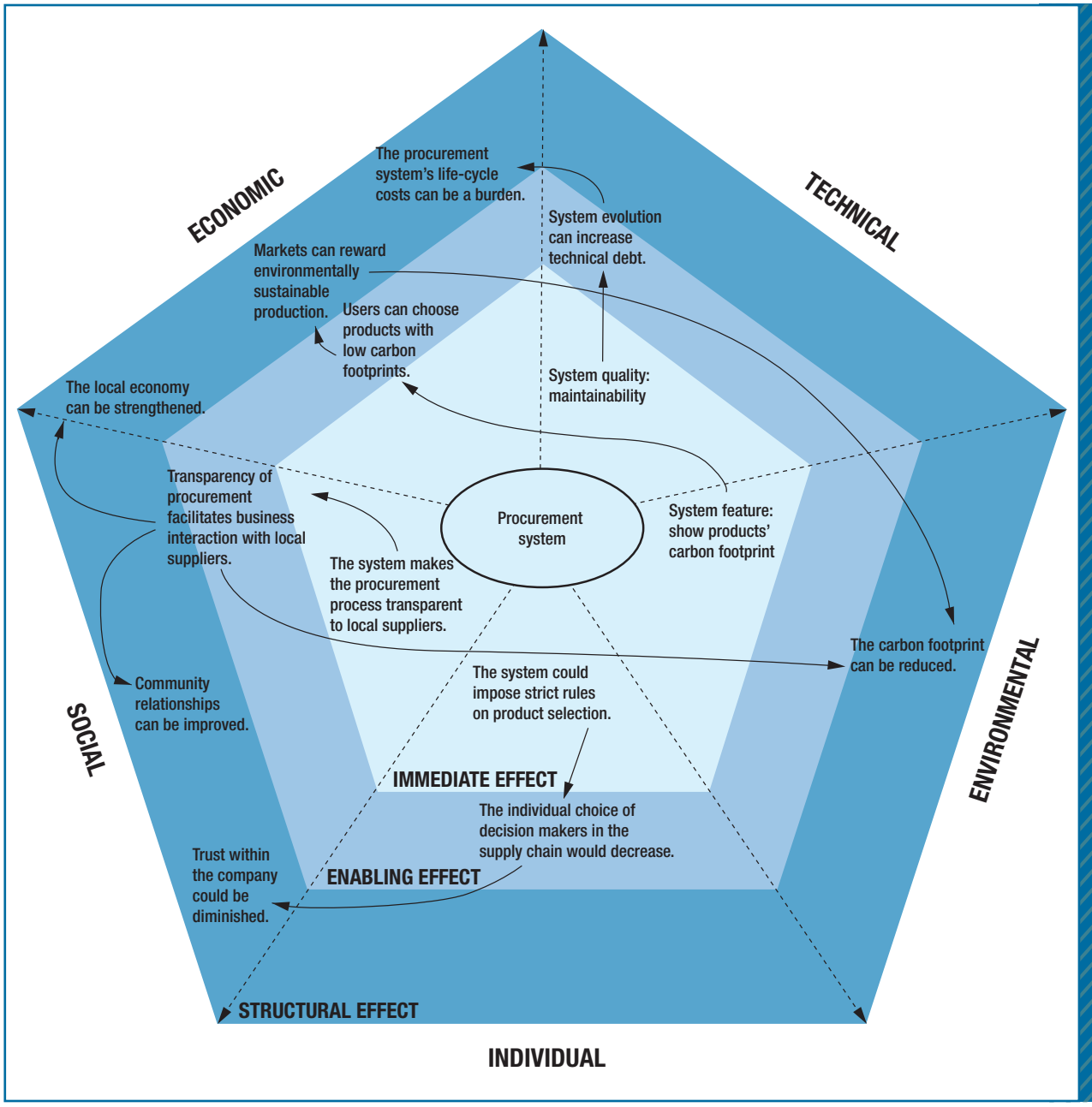


**FIGURE 1.** Selected immediate, enabling, and structural effects of the procurement system in the five sustainability dimensions. The diagram supports interactive collaboration among stakeholders to discover, document, and validate the system's potential effects.

abound, ranging from architectural design patterns to documentation guidelines. Yet, because applying these techniques often involves an up-front investment of effort, it occurs only when a longer life expectancy of a system is recognized and expressed. On the other hand, a stated requirement for which no technique yet exists will lead to an identified gap in technological ability. This means that in practice, systemic changes to the activities in Table 1 will dominate the effects of whatever techniques we develop to support these activities.

So, requirements engineers play a key role in sustainability. As "sustainability engineers," they go beyond a narrow system perspective and follow an interdisciplinary,

TABLE 1

Table 1. Software engineering practices for sustainability.*

| Task | Standard current practice | Focus of future practice |
|---|---|---|
| Mind-setting | The world is a puzzle, and we should solve the problem. | The world is complex, and we should first understand the dilemmas. |
| Determination of the project objective and the system purpose, boundary, and scope | Focus on the immediate business need and key system features. Don't question the project's or system's purpose. | Emphasize how the project can affect sustainability in all dimensions. Strive to advance sustainability in multiple dimensions simultaneously. Experiment with different system boundaries to understand the alternative impacts. |
| External constraint identification | See constraints as imposed by the direct environment of the system and its technical interfaces. Minimize the constraints considered, but include legal, safety, security, technical, and business resources. | See constraints in each dimension as opportunities. Look for constraints from additional sources, starting with company corporate-social-responsibility policies, legislation, and sustainability standards. |
| Stakeholder identification | Minimize the number of stakeholders involved, and focus on those who have influence. Focus on internal stakeholders, and exclude unreachable stakeholders. | Maximize stakeholder involvement in an inclusive perspective integrating external stakeholders, and involve those who are affected. Assign a dedicated role to be responsible for sustainability, and introduce surrogate stakeholders to represent outside interests. |
| Success criteria definition | Focus on the financial bottom line at project completion. Measure the business outcome and financial return on investment. | Focus on advancing multiple dimensions simultaneously, including financial aspects, and take into account that most effects occur after project completion. |
| Requirements elicitation | Focus on the features and immediate effects the stakeholders want. | Help the stakeholders understand the system's enabling effects. Use creativity techniques and long-term scenarios to forecast the potential structural impact. |
| Risk identification | Identify risks that threaten timely project completion within the budget. | Include the effects on the system's wider environment. Include enabling and structural effects and risks that can develop over time. |
| Tradeoff analysis | View tradeoff analysis as a prioritization and selection problem, and let the key stakeholders decide. | Strive to transform sustainability tradeoffs into mutually beneficial situations. Ensure that a wider range of stakeholders (or their surrogates) discuss sustainability tradeoffs. |
| Go/no-go decision | Base the decision on feasibility, financial costs and benefits, and risk exposure to project participants—that is, internal stakeholders. | This continues to be an internal business decision but is documented to show to external audiences that it took into account sustainability indicators and enabling effects. The decision is based on a consideration of positive and negative effects in all five dimensions. |
| Requirements validation | Let key stakeholders verify that their interests are captured. | Ensure broad community involvement focused on understanding effects. |
| Project completion | Verify whether success criteria are met on the completion date. After that, focus on maintenance and evolution. | Evaluate the effects in all five dimensions over a certain time frame after completion, aligned with the expected timescale of effects. |
| Requirements documentation | Current templates ignore long-term effects and sustainability considerations. | Templates require information about sustainability as a design concern and support analysts with checklists. |

** For a description of the dimensions mentioned in the table, see the section "Sustainability in Software Engineering."*

systems-oriented, stakeholder-focused approach, supported by higher management and executives. Their task is to understand the nature of software-intensive systems and the impact those can have on their social, technical, economic, and natural environments and the individuals in those environments.

This responsibility is reflected in the new UK Standard for Professional Engineering Competence, which specifies that engineers are to "act in accordance with the principles of sustainability, and prevent avoidable adverse impact on the environment and society."[11] It's up to SE curricula developers to equip

future software engineers with the competences required to simultaneously advance goals in all five dimensions, beyond the technical and economic.

For a long time, concerns about such effects have taken a backseat in SE, but this is changing as standards are being adjusted. For example, the working group WG42 on ISO/IEC 42030 (Architecture Evaluation) is discussing energy efficiency and environmental concerns at the software architecture level. In addition, the IEEE P1680.1 Standard for Environmental Assessment of Personal Computer Products is being revised.

Although these steps are important, a full consideration of all five sustainability dimensions is needed on the level of quality models, system documentation templates, and the analysis of systemic effects throughout system life-cycle stages. Requirements engineers will often be responsible for introducing relevant standards in each of the five dimensions into the elicitation and specification process. To support this, revisions of the ISO 25000 series should incorporate sustainability considerations related to software systems' quality attributes. In addition, ISO 29148 should acknowledge the importance of system characteristics beyond interaction with human users and encourage consideration of the systemic effects of software systems in RE.

Software's critical role in society demands a paradigm shift in the SE mind-set. Sustainability design emphasizes an appreciation of wicked problems over a focus on puzzles and pieces, systems thinking over computational problem solving, and an integrated understanding of systems over a divide-and-conquer approach to systems analysis.

Although these challenging shifts won't come easy, taking such perspectives provides an opportunity to stand out, an invitation to innovate, and an occasion for software engineers and companies to distinguish themselves with a unique selling point in a competitive market. We also have the opportunity to help shape broader sustainability policy. A shift to a sustainable society requires large-scale change both in government policy and in engineering and business practice; neither on its own will suffice. But regulatory change is much easier if it builds on established best practices, so software practitioners must take the lead.

If you agree that we, as software engineers, have a responsibility for the long-term impact of the systems we design, the sustainability design principles provide an opportunity to get started. We can and should start now, and practitioners can lead the way. We need to collect experiences in applying sustainability principles in SE and learn from the process. An important way to make this vision of software as a force for sustainability a reality is by cooperation between industry and academia.

Successful collaborations to integrate sustainability concerns into established practices can significantly and positively influence the long-term effects of the systems we design. To facilitate this, we must do three things.

First, we must identify and tackle causes of unsustainable software design. For this, industry can invite academics to research, analyze, and reengineer their current development processes and practices for improved sustainability.

Second, we must develop exemplar case studies that demonstrate the benefits of sustainability design in SE. For this, early adopter industrial collaborators can partner with academics to apply research findings such as those summarized in Table 1 and report on longer-term results.

> Software's critical role in society demands a paradigm shift in the software engineering mind-set.

Finally, we must build competences in the theory and practice of sustainable design into the training of all software engineers. Industry can make the demand for software practitioners trained in sustainability principles explicit by requiring specific competences from potential employees. Researchers and educators should develop improved curricula that incorporate sustainability principles and ensure that future software professionals possess the competences needed to advance sustainability goals through SE.

Let's get started.


## Acknowledgments

This research is supported by the Deutsche Forschungsgemeinschaft project EnviroSiSE (PE2044/1-1); FAPERJ (210.551/2015); CNPQ (14/2014); NSERC (RGPIN-2014-06638); the European Social Fund; the Ministry for Science, Research, and the Arts Baden-Württemberg;

## ABOUT THE AUTHORS

**CHRISTOPH BECKER** is an assistant professor at the University of Toronto, where he leads the Digital Curation Institute, and a senior scientist at the Vienna University of Technology. His research focuses on sustainability in software engineering and information systems design, digital curation and digital preservation, and digital libraries. Becker received a PhD in computer science from the Vienna University of Technology. Contact him at christoph.becker@utoronto.ca.

**STEFANIE BETZ** is a senior research scientist in the Karlsruhe Institute of Technology's Department of Applied Informatics and Formal Description Methods. Her research centers on sustainable software and systems engineering, particularly from the perspective of requirements engineering and business process management. Betz received a PhD in applied informatics from the Karlsruhe Institute of Technology. Contact her at stefanie.betz@kit.edu.

**RUZANNA CHITCHYAN** is a lecturer in the University of Leicester's Department of Computer Science and a member of the Centre for Landscape and Climate Research. Her research centers on requirements engineering and architecture design for software-intensive sociotechnical systems and sustainability. Chitchyan received a PhD in software engineering from Lancaster University. Contact her at rc256@leicester.ac.uk.

**LETICIA DUBOC** is a lecturer in the State University of Rio de Janeiro's Department of Computer Science and an honorary research fellow at the University of Birmingham. Her research focuses on software system sustainability and scalability, particularly from the perspective of requirements engineering and early analysis of software qualities. Duboc received a PhD in computer science from University College London. Contact her at leticia@ime.uerj.br.

**STEVE M. EASTERBROOK** is a professor in the University of Toronto's Department of Computer Science and a member of the School of the Environment and the Centre for Global Change Science. His research focuses on climate informatics—specifically, applying computer science and software engineering to the challenge posed by global climate change. Easterbrook received his PhD in computing from Imperial College London. Contact him at sme@cs.toronto.edu.

**BIRGIT PENZENSTADLER** is an assistant professor of software engineering at California State University, Long Beach. Her research focuses on software engineering for sustainability and resilience; she leads the university's Resilience Lab. Penzenstadler received a habilitation in environmental sustainability in software engineering from the Technical University of Munich. Contact her at birgit.penzenstadler@csulb.edu.

**NORBERT SEYFF** is a professor in the School of Engineering and the Institute of 4D Technologies at the University of Applied Sciences and Arts Northwestern Switzerland and a senior research associate in the University of Zurich's Department of Informatics. His research focuses on requirements engineering and software modeling, particularly on empowering and supporting end-user participation in system development. Seyff received a PhD in computer science from Johannes Kepler University Linz. Contact him at norbert.seyff@fhnw.ch.

**COLIN C. VENTERS** is a senior lecturer in software systems engineering at the University of Huddersfield. His research focuses on sustainable software systems engineering from a software architecture perspective for presystem understanding and postsystem maintenance and evolution. Venters received a PhD in computer science from the University of Manchester. Contact him at c.venters@hud.ac.uk.

and the Vienna Science and Technology Fund (WWTF) through project BenchmarkDP (ICT2012-46). Special thanks to our friend and colleague Sedef Akinli Kocak, a PhD researcher at Ryerson University, for her contributions to this article.

## References


1. H. Koziolek, "Sustainability Evaluation of Software Architectures: A Systematic Review," *Proc. Joint ACM SIGSOFT Conf.—QoSA and ACM SIGSOFT Symp.—ISARCS on Quality of Software Architectures—QoSA and Architecting Critical Systems—ISARCS* (QoSA-ISARCS 11), 2011, pp. 3–12.
2. J.A. Tainter, "Social Complexity and Sustainability," *Ecological Complexity*, vol. 3, no. 2, 2006, pp. 91–103.
3. B. Penzenstadler et al., "Safety, Security, Now Sustainability: The Nonfunctional Requirement for the 21st Century," *IEEE Software*, vol. 31, no. 3, 2014, pp. 40–47.
4. J. Elkington, "Enter the Triple Bottom Line," *The Triple Bottom Line: Does It All Add Up? Assessing the Sustainability of Business and CSR*, A. Henriques and J. Richardson, eds., Earthscan, 2004, pp. 1–16.
5. "Who Lives the Longest? Busting the Social Venture Survival Myth," E3M, 2014; http://socialbusinessint.com/wp-content/uploads/Who-lives-the-longest_-FINAL-version2.pdf.
6. C. Bomfim et al., "Modelling Sustainability in a Procurement System: An Experience Report," *Proc. IEEE 22nd Int'l Conf. Requirements Eng.* (RE 14), 2014, pp. 402–411.
7. H.W. Rittel and M.M. Webber, "Dilemmas in a General Theory of Planning," *Policy Sciences*, vol. 4, no. 2, 1973, pp. 155–169.
8. S. Easterbrook, "From Computational Thinking to Systems Thinking: A Conceptual Toolkit for Sustainability Computing," *Proc. 2nd Int'l Conf. Information and Communication Technologies for Sustainability*, Atlantis Press, 2014; doi:10.2991/ict4s-14.2014.28.
9. J.A. Klein, "A Reexamination of Autonomy in Light of New Manufacturing Practices," *Human Relations*, vol. 44, no. 1, 1991, pp. 21–38.
10. S. Betz et al., "Sustainability Debt: A Metaphor to Support Sustainability Design Decisions," *Proc. 4th Int'l Workshop Requirements Eng. for Sustainable Systems* (RE4SuSy 15), 2015; http://ceur-ws.org/Vol-1416/Session2Paper4.pdf.
11. *UK Standard for Professional Engineering Competence (UK-SPEC)*, Engineering Council, 2014.