

Software Engineering for Sustainability

Find the Leverage Points!

Birgit Penzenstadler, California State University, Long Beach

Leticia Duboc, La Salle—University Ramon Llull

Colin C. Venters, University of Huddersfield

Stefanie Betz, Furtwangen University

Norbert Seyff, University of Applied Sciences and Arts
Northwestern Switzerland

Krzysztof Wnuk, Blekinge Institute of Technology

Ruzanna Chitchyan, University of Bristol

Steve M. Easterbrook and Christoph Becker, University of Toronto

// Software is a pervasive driver of change in society. Therefore, software professionals need to take a systems perspective—and identify leverage points to understand the role of software for transformational change toward sustainability. //

SOFTWARE ENGINEERING HELPS

deliver software systems that can enable humanity to reach new levels of prosperity. That experience in building complex, interdependent, and globally distributed systems can also be leveraged for sustainability challenges. Humanity faces a number of global, interdependent, and complex challenges that present a risk to societies, including climate change, large-scale involuntary migration, and poverty.¹

As software professionals, we can contribute to sustainability through the software systems that we engineer, and it is our social responsibility to do so.² But sustainability problems are complex system problems (see the sidebar “Sustainability”). How can we understand the complex dynamics that arise in the interaction within multifaceted social, economic, or ecological systems? One approach to identifying successful sustainability interventions is to consider *leverage points* (LPs)—locations within a system where a small change in one aspect can result in significant system-wide changes.³

This article suggests that LPs can help software engineers to address sustainability challenges by offering insights on possible transformation mechanisms or ways to find alternatives. While LPs will not tell us exactly how to act on sustainability challenges, they provide an analysis tool to help practitioners to identify elements that can bring about effective change at different levels, for a (software) system and the wider system it resides in. As sustainability is a crosscutting (orthogonal) concern, LPs are beneficial because they enable intervention on different levels.

We use the example of the UK public-transportation system⁴ to



illustrate how LPs can contribute to software engineering for sustainability.

The UK Public-Transportation System

Existing transportation systems are large contributors to greenhouse gas emissions and poor urban air quality, which contribute toward health issues and general environmental unsustainability. Operations researchers have been developing systems to improve transportation for many decades using linear programming and simulation systems, while new approaches to data science offer a future vision of a smart transportation system based on IT-supported movement of people and goods.⁴ However, the factors that impede sustainability are complex. Figure 1 shows the UK transportation system in the context of its surrounding systems, using a stock-and-flow model annotated with causal feedback loops.⁵

As in any complex system, this example is embedded within a set of assumptions—i.e., a paradigm. In this case, it is the shared beliefs that people need transportation, that they have some choice over which mode to use, and that government spending and provisioning of bus and road capacity should support this choice. The system aims to achieve certain goals, while obeying a set of rules. The main goal of our example system is to transport people, with the rules given by the existing infrastructure.

A system-dynamics model captures only a partial view of a system but helps build a more holistic understanding by looking at chains of cause and effect to identify points through which desired changes could be reinforced or undesired changes



SUSTAINABILITY

The *Oxford English Dictionary* defines sustainability as “the capacity to endure.”¹⁹ The Brundtland Commission defined sustainable development as “meeting the needs of the present without compromising the ability of future generations to meet their needs.”²⁰ However, to understand the broader sustainability issues, we must ask which system to sustain, for whom, over which time frame, and at what cost.²¹ This involves five interrelated dimensions:¹⁴

- The *individual dimension* covers individual freedom and agency, 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. 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, ranging from immediate waste production and energy consumption to the balance of local ecosystems and climate change concerns.

prevented. Within the system-dynamics perspective, a system is viewed as a set of *stocks* (any quantity that accumulates or depletes over time), such as the number of private vehicles. The level of a stock can be changed via *flows*, where the flows define a rate of change of the given stock. Stabilizing stocks are known as *buffers*. The intensity of a flow can be influenced through parameters—for example, governments can set congestion fees or adapt taxes. The larger the stock with respect to the rate of its flows, the more stable it is (e.g., a large public-transportation network is more likely to create a more stable revenue).

The change in stocks due to flows is often nonlinear due to feedback

loops, which occur when a changing level of a stock or a flow creates a circular chain of cause and effect that eventually influences the original stock or flow. For example, see B1 in Figure 1. If buses become frequent and uncrowded, more people are likely to switch from using their car to buses, increasing the demand for buses, making them more crowded and hence less attractive again. B1 is a *balancing feedback loop*, as it counteracts the original change.

However, the more people switch from cars to buses, the greater the revenue generated from bus pricing (see R1 in Figure 1). If this revenue is used to increase the fleet and, consequently, the availability of buses,

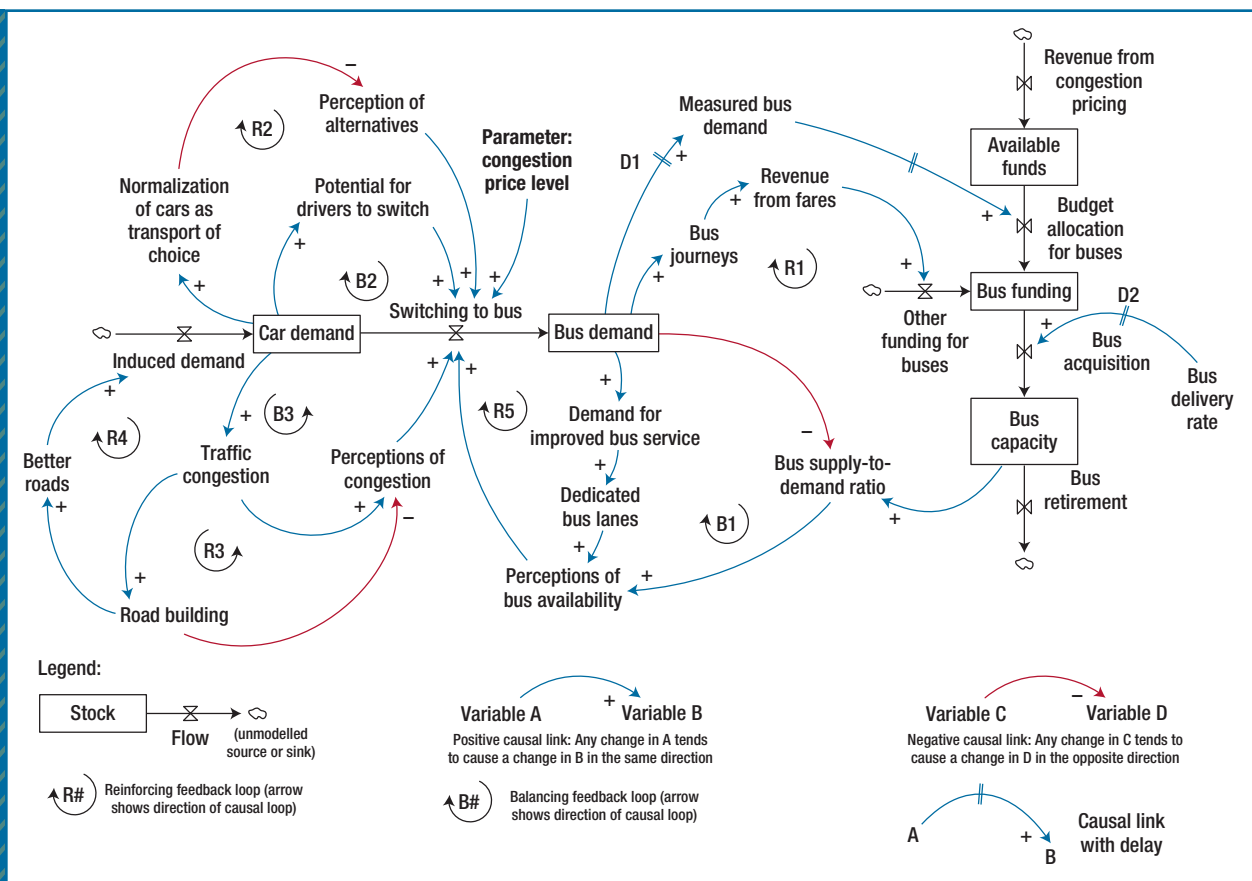


FIGURE 1. A causal loop diagram of transportation showing contextual factors for mode switching, such as traveler perceptions, funding levels, and relative demand.

it can encourage even more people to switch from cars to buses. R1 is a *reinforcing feedback loop* because it pushes a change even further. Reinforcing feedback loops can spiral out of control, but they eventually meet some bound; e.g., when everybody uses buses, the demand cannot increase further (see B2). However, another balancing feedback loop may intervene before that, because as soon as people perceive less congestion, they tend to switch back to using cars again (see B3).⁶

The effects of flows on stocks may not be immediate. For instance, it may take time to gather information about changes in demand for public transportation (see delay

D1 in Figure 1), and even longer to adjust the supply of buses (see D2) to achieve the goal of increased public-transportation usage. The length of the delay affects the stability of the system.

The structure of stocks and flows also has a huge effect on the system's behavior. For example, if revenue from congestion charges (whereby drivers must pay to drive through highly congested areas—see the top middle of Figure 1) flows into more investment in more public transport, more people may choose to switch from cars to public transport. But if it flows to more road building, it may have the opposite effect. Further concepts such as the economics of supply and demand, time

cost, and rebound effects are not addressed here. For further details, see “Integrating Renewable Energy Forecast Uncertainty in Smart-Charging Approaches for Plug-In Electric Vehicles”⁷ and “Telecommunications and Travel: The Case for Complementarity.”⁸

Creating Leverage through Software

How can software effect change in its wider environment? Donella Meadows proposed a list of 12 ways of intervening in a system (any type of system), as an invitation to think more broadly about change³ (see the sidebar “Leverage Points”). We discuss these 12 LPs in four clusters⁹ in



LEVERAGE POINTS

Donella Meadows stated that “Leverage points are places within a complex system (a corporation, an economy, a living body, a city, an ecosystem) where a small shift in one thing can produce big changes in everything.”³ Table A lists the leverage points (LPs) in increasing order of effectiveness according to Meadows. While all LPs can bring about change, the later ones are more likely to create significant changes to the system behavior but may also require more effort to implement. Meadows’s LPs refer to any kind of change, whether enabled by software or not. In the main article, we use them as an analysis tool for exploring how software can trigger broader changes in societal systems. However, they are hard to identify and act on—they are not a silver bullet.

Table A. Leverage points.³

| Leverage point | Description |
|----------------|---|
| LP 12 | Constants, parameters, and numbers. Tweaking parameters allows change to the intensity of the flows in systems but rarely alters the underlying dynamics. |
| LP 11 | The sizes of buffers and other stabilizing stocks, relative to their flows. Stabilize a system by adjusting the capacity of its buffers, and make it more efficient by optimizing the flow. |
| LP 10 | The structure of material stocks and flows (such as transportation networks and population age structures). Physical structure is crucial in a system but often hard to change; therefore, the leverage point is in proper initial design. |
| LP 9 | The lengths of delays, relative to the rate of system change. A system cannot respond to short-term changes when it has long-term delays. |
| LP 8 | The strength of balancing feedback loops, relative to the impacts they respond to. Balancing feedback loops help systems to self-correct by monitoring and adjusting according to the system goal. |
| LP 7 | The gain around reinforcing feedback loops. Reinforcing feedback loops can be sources of system instability or mechanisms to amplify desired change, so adjusting their strength affects how the system responds to change. |
| LP 6 | The structure of information flows. This can create a new feedback loop that was not there before. Altering the structure of information flows enables more agency by users. |
| LP 5 | The rules of the system, including incentives, punishments, and constraints. Social rules include constitutions, laws, standards, policies, and incentives. Changing the rules of a system can change the behavior of the society under them. |
| LP 4 | The power to add, change, evolve, or self-organize system structure. In biology, this is called evolution; in society, we call it empowerment. In systems terms, it is called self-organization, the strongest form of system resilience. |
| LP 3 | The goals of the system. Changing the goal of a system is a powerful strategy to effect change but can be hard to achieve. |
| LP 2 | The mind-set or paradigm out of which the system arises. Paradigms are a shared set of deep beliefs about how the world works. They are the hardest to change in a system, as society will fiercely resist any challenges to its paradigms. |
| LP 1 | The power to transcend paradigms. This final and most effective LP is about being unattached to existing paradigms; there is no certainty in any particular worldview. |

increasing order of the likely magnitude of their effect:

- changing the metabolic structure of the system,
- changing the feedback loops,
- transformational change, and
- changing the intent of the system and stakeholders.

For a synopsis of the role of software in relation to LPs in our UK transportation example, see Table 1. (Throughout the rest of the article, “LP 1,” “LP 2,” and so on indicate

Table 1. The role of software in our example and the impact per leverage point (LP) cluster.*

| Cluster | Objective | Role of software | Affected element in the traffic example |
|--|--|---|---|
| Changing the metabolic structure (LP12, LP11, and LP10) | Optimize the bus fleet to balance the number of passengers and buses needed. | Monitor and analyze the transportation network and road traffic to understand commuter patterns and the frequency and type of failures affecting buses, and then accordingly <ul style="list-style-type: none"> • adjust the number of buses to increase flow and • calculate the size of the buffer of spare buses. | Buffer: A number of spare buses is maintained that allows increased flow (frequency) when necessary. |
| Changing the feedback loops (LP9, LP8, and LP7) | Maintain a positive perception with respect to bus availability; e.g., reduce the perception of overcrowding and bus shortages. | Monitor traffic development over time and identify commuting patterns to reduce overcrowding and bus shortages by <ul style="list-style-type: none"> • redeploying buses to particularly busy routes to reduce overcrowding and • designating additional lanes as bus-only lanes when traffic congestion increases. | Balancing feedback loop: A positive perception is maintained with respect to bus availability via <ul style="list-style-type: none"> • an increase in the total number of buses and • an increase in the ratio of bus supply to demand. |
| Transformational change and self-adaptation (LP6, LP5, and LP4) | Alter the structure of information flows for bus schedules and actual departure times to increase the convenience of bus use and total demand. | <ul style="list-style-type: none"> • Inform the users of the time they need to be at a bus stop by providing information on the current location of the buses and their impending arrival time (e.g., via apps). • Instruct the maintenance team on preventing frequent failure types. | Information flows: Up-to-date bus schedules are sent to app users. |
| Changing the intent of the system and stakeholders (LP3, LP2, and LP1) | Provide public transportation as a “free” service that is paid for by the government (a right to public mobility, like public health) to reduce pollution, improve health benefits, and increase personal freedom and opportunities. | <ul style="list-style-type: none"> • Ensure that transportation meets the whole range of needs by better supporting the choice of transportation modes according to calendar synchronization and traffic. • Provide a traffic system that responds to the analysis of data in real time (and includes renewable-energy forecasting). • Support the elderly (accessibility). • Provide info about the use of public transportation, and show how this contributes to environmental health. | Goals for commuting: The whole transportation system as well as its societal context is affected as the underlying structure changes. |

* For explanations of LP1 through LP12, see the sidebar “Leverage Points.”

the LPs in the “Leverage Points” sidebar.)

Changing the Metabolic Structure of the System (LP12, LP11, and LP10)

These LPs fine-tune the way a system operates, without changing its nature. This includes changing the value of parameters, sizes of buffers, and the material stocks and flows. Such changes can help stabilize and optimize the use of resources. However, such changes have relatively

low leverage because they rarely alter the cause-and-effect loops that shape the dynamic behavior of a system. So, they often fail to initiate broader change and sometimes end up entrenching current behaviors.³

For example, we can view the public-transportation network of a city as a structure of material stocks and flows with many parameters, including pricing mechanisms, road capacity, bus frequency, and so on. Some cities use congestion charges,

with higher fees for larger engines that produce more emissions. This can encourage commuters to choose public transportation rather than pay the congestion charges.

Software is often the critical enabling technology that offers new ways of setting, changing, and monitoring parameters to adjust the flows within a system in real time—for example, by monitoring which cars enter the zones with congestion charging, calculating based on peak

times, etc. Often, adjusting parameters is insufficient to bring about lasting change. For example, simply increasing the congestion charge within a city will not resolve the congestion problem if commuters believe that they have no other convenient alternatives (see R2 in Figure 1). The feedback loops that cause people to prefer to drive remain unaltered.

Software systems also offer new ways to analyze and optimize stocks, flows, and buffers within the existing transportation network. For example, by continuously monitoring commuter patterns, software systems can adjust the frequency of buses to increase the flow, or recommend higher-capacity buses on certain routes to provide a larger buffer that reduces overcrowding. In the long term, data analytics can help to optimize the structure of the transportation network, by recommending changes to existing bus routes.

But when we rush to apply software technology in these ways, we risk missing the bigger picture. For example, by optimizing the flow of traffic through intersections, we may inadvertently strengthen the existing feedback loop that encourages people to drive more when they perceive there to be less congestion (see R3 in Figure 1). By implementing software solutions to tweak pricing strategies, we may divert attention from issues such as the lack of capacity in public transit, which act as a much bigger barrier. This way, by applying simplistic solutions to low-leverage changes, we may end up making the sustainability problem worse.¹⁰

Changing the Feedback Loops (LP9, LP8, and LP7)

This cluster of LPs addresses the power that balancing and reinforcing feedback loops can have on the

stability of systems. It also encompasses the consequences delays can have on system change.

In transportation, reinforcing feedback loops often push the system further away from sustainability. For example, over the long term, better roads cause more people to buy and use cars, which increases the demand for more and better roads (see R4 in Figure 1). Such reinforcing loops are powerful, as they amplify change within a system. As Meadows pointed out, “A system with an unchecked [reinforcing] loop ultimately will destroy itself.”³

Software can help to improve our understanding of such reinforcing loops and their consequences. For example, computer simulations provide an opportunity to explore policies that can break such loops, by testing the dampening effect of road pricing or the deliberate reduction of road capacity through pedestrianized streets. Software also offers new opportunities to weaken such loops—for example, by connecting people more readily to car-pooling and car-sharing services, to slow the growth of car ownership.

However, not all reinforcing feedback loops are undesirable. A similar loop can accelerate switching to buses instead of cars. For example, priority bus lanes can improve the perception of buses as a convenient alternative and encourage more people to switch to buses, who then demand even more such improvements (see R5 in Figure 1). For cities struggling with traffic congestion and poor air quality, this loop is beneficial if it leads to fewer motor vehicles. Social media apps can play a significant role in amplifying such a loop, as they allow people to encourage others to switch.


Balancing feedback loops regulate and stabilize a system by pushing back against change. As with reinforcing loops, this might be good or bad, depending on the nature of the change. The balancing loops shown in Figure 1 all tend to work against the goal of getting car users to switch to buses. They suggest that once enough people switch to buses, the increased crowding on buses and reduced traffic congestion will cause people to perceive driving as more convenient again, thus slowing or reversing the change. If software solutions to traffic management ignore such feedback loops, they are unlikely to be successful. But software also provides the opportunity to reduce the impact of these loops—for example, by monitoring demand and dynamically re-deploying buses to reduce crowding, and by identifying opportunities to create more bus lanes when the traffic congestion eases.

Delays within a system are a common source of oscillations, which occur when a problem builds up because corrective action arrives too late, and again when a corrective action overcompensates because it is applied for too long. For example, in Figure 1, delays in securing funding and procuring new buses (see D2 in Figure 1) can undermine a strategy of getting people to switch, if they cause large swings in the quality of service such that people start to lose faith in it. Well-designed software systems can greatly reduce these problems by improving access to data and shortening response times, but only if the software designers understand the impact of such delays.

Transformational Change (LP6, LP5, and LP4)

These LPs encompass transformational changes in systems—namely,

ACTION POINTS FOR APPLYING LEVERAGE-POINT THINKING IN SOFTWARE ENGINEERING



Two basic questions arise during the development of socio-technical systems: “Are we building the right system?” and “Are we building the system right?” The first question is often interpreted narrowly within the context of the business problem the software system is trying to solve. However, can any system not supporting the sustainability of our society be “the right system”? What should software engineers do to ensure they are building a system that also contributes to sustainability?

Using stock and flow diagrams and keeping in mind the effectiveness of the different leverage points during the requirements analysis, collaborate with domain experts to answer the following questions:

- What stocks and flows are affected by the system (e.g., energy, natural resources, or the supply of goods)? Can software stabilize them (e.g., by optimizing buffers, reducing delays for adjusting quantities, making the state of the system known, or monitoring and adjusting parameters)?
- What circular chains of cause and effect exist in the system? How do they reinforce or balance changes

in the stocks and flows? Can your software initiate or disrupt a feedback loop, and if so, how (e.g., by providing information, counteracting the original change, or detecting when the system threatens to go out of bounds)? Use simulations.

- What is the structure of the information flows of the system? Can your software make missing information available to stakeholders?
- Are the goals of the system appropriate (e.g., seek to improve social connectedness, to reduce waste production, or to simplify logistics)? Can these goals be negotiated? Will your software encourage society to reflect on and potentially change the goals and paradigms of systems? Are the means by which goals are achieved by the system pertinent? Can your software enable different means to achieve the goals?
- In the cases above, where exactly will your software intervene? What will be the likely results of such intervention? Has the customer been made aware of this?

Finally, decide where to apply your skills, and take responsibility for the software you build.

the structure of information flows, rules, and self-adaptation. Such changes have high leverage, as they can sweep away existing feedback loops (or rebound effects) and generate entirely new system behaviors.

Changing the information flows within a system can have a dramatic effect. For example, real-time information on bus schedules and actual arrival times may increase user satisfaction. Changing the rules can be equally effective; for example, providing free bus rides for children may instantly bring a whole new group of users. And self-adaptation empowers learning within a system—for example, when revenue-raising powers are

devolved to cities and towns, they can develop solutions tailored to the local situation.

Ride-brokering systems, such as Uber, provide an example of a software-driven intervention that hits all three of these LPs. These systems change the information flows by connecting people who need transportation with those who can supply it. They change the rules by side-stepping existing regulations around licensing and safety regulations for taxis. They provide a kind of self-adaptation, as users can (somewhat) negotiate service-level agreements among themselves. The current uproar in cities across the world in reaction to those

services attests to their high leverage potential.

But whether this change is for the better or worse may depend on whom you ask.¹¹ Those whose interests were protected by the old system (taxi owners and people who value privacy) regard this as negative, whereas affluent urban users often love it. While such systems are often held up as models of how software can bring about disruptive innovation, few people stop to consider the impact on sustainability.

We outline how software engineers can do that in the sidebar “Action Points for Applying Leverage-Point Thinking in Software

Engineering.” Maybe ride-hailing services (along with self-driving cars) are not radical enough as changes.¹² Because they still emphasize the car as the dominant mode of transportation, they lock us into highly energy- and material-intensive forms of personal transportation that are ill suited to dense urban settings and do little to reduce our environmental footprint.¹³

Changing the Intent of the System and Stakeholders (LP3, LP2, and LP1)

The most radical system changes tend to occur when we change our goals or the mind-set (“paradigm”) that shapes them, or when we learn to transcend paradigms and see the world from multiple perspectives. Accordingly, these kinds of change have the most leverage but are often much harder to bring about.

Our goals are important because they constrain how we think and set our expectations for success criteria. For our transportation example, the goal of switching people from cars to buses might be futile; perhaps a better goal is to reduce the need to commute overall. Software could help in planning walkable communities with regard to city layout, optimal distribution of living and working areas, etc. This needs to be discussed with different stakeholders, and software engineers can clarify different design options. Instead of optimizing existing operations, focus on longer-term goals. What would the sustainable city of the future look like, and what kinds of software are needed to implement it? We need to look beyond clients’ initial ideas for what they want and to help them identify their real goals and the kinds of software capability that would help meet them.¹⁴

Paradigm changes are harder and are unlikely to be driven by software

solutions alone. However, software can support such a shift. With society on the verge of a massive technology transition, driven by a push to a zero-carbon economy,¹⁵ our paradigms for *why* we develop software may need to change along with the *how*. For example, instead of building software to automate or optimize existing transportation systems (with unsustainable behaviors driven by their feedback loops), we should focus on our ethical responsibilities to society and seek opportunities to create software that changes how we perceive the transportation system, and hence how we think about sustainability.

Software is deeply ingrained in our society. By nature, software systems provide ubiquity, fast distribution, huge computing power for data analysis and simulation, immediacy of computational results, and a potentially global effect. As such, software systems can be drivers of change.¹⁶ However, if we fail to understand them as drivers of change, we may miss much of this potential. A holistic analysis of the systems in which our software will be deployed provides an important starting point for understanding the set of LPs we have access to, and how to deploy them (see the sidebar “Evidence for the Impact of Leverage Point Analysis”).

Measuring the impact of applying LPs initially requires a higher level of abstraction than the metrics typically used in software development. Instead, we can start with a triangulation as proposed in “Soft Systems Methodology: A Thirty Year Retrospective”¹⁷ that selects metrics to assess the efficiency, effectiveness,

and efficacy of a specific intervention. These subsequently have to be refined for software systems.

The important paradigm shift is for software professionals to take more responsibility for the broader social and environmental impacts of software technology, by thinking about LPs and how to make use of them in software design for sustainability.¹⁸

On a societal level, these LPs challenge us to engage in a (philosophical and practical) discussion of whether we, as a society, have a goal other than a specific short-term quality of life, and whether the technology we develop locks us into this short-term thinking or helps bring about a sustainable society. Such discussion, also taking into account political forces, is paramount for influencing the decision makers shaping our systems.

In our examples, software is the critical enabling technology that allows us to exploit an LP. There are many challenges in doing so—inter alia, understanding LPs, seeing the bigger (complex) picture, identifying the information flows, and recognizing the extent of feedback loops. However, the more fundamental the change we seek, the less we can expect software *per se* to bring it about.

Enabling these LPs goes beyond software engineering. It is about the personal choice of each of us to commit to make the world a better place facilitated by software. Software engineers need to be aware of the power of software systems as a transformational force in society and the significant impact that their designs can have. As software engineers, we may perceive our responsibilities to be limited to our customers’ immediate concerns. But, as experts in

EVIDENCE FOR THE IMPACT OF LEVERAGE-POINT ANALYSIS

Leverage point (LP) analysis has had a tremendous impact in many areas—e.g.,

- leveraging feedback loops in the development of urban policy in response to climate change in Australia,²²
- leveraging a mind-set change and goal setting (as well as other LPs) with a focus on children's health to improve healthcare systems at six US sites during a 2004 to 2009 study,²³ and
- using metabolic-structure LPs in global simulations to increase food security while lowering the environmental impact.²⁴

Furthermore, John Sterman documents the following three major case studies.²⁵

GENERAL MOTORS (MID-1990S)

GM analyzed the impact of leasing on new-car sales, particularly the rise in the availability of high-quality off-lease used vehicles. The system-dynamics analysis revealed two key LPs: a policy change toward longer lease terms and a new information flow, to collect data on changes in consumer choice. Other LPs included changing the incentives for managers within GM to do full profit-and-loss accounting of their leasing policies. Applying these LPs allowed GM to weather the slump in used-car prices in 1997 much better than its competitors, who eventually also adopted GM's approach.

INGALLS SHIPYARDS (LATE 1970S)


In a lawsuit against the US Navy, Ingalls claimed that design changes imposed by the Navy on a fleet of ships Ingalls was contracted to build would lead Ingalls to lose \$500 million on the contract. A systems analysis identified a set of feedback loops around the cost of rework and worker retention, showing that these feedback loops were the key LP for managing cost overruns and that the problem was not poor project management (as the US Navy had argued). The analysis allowed Ingalls to make a rapid out-of-court settlement and led to long-term improvements in how the US Navy manages design changes in its contracts.

DUPONT (MID-1990S)

After a long history of cost-cutting measures, DuPont found its maintenance costs had skyrocketed. An LP analysis revealed a major factor was the resulting steady rise in the stock of latent defects in DuPont's equipment. The analysis suggested three key LPs:

- a change in goals from defect correction to defect prevention,
- a weakening of the feedback loop in which cost-cutting leads to increased breakdowns and a "firefighting" mentality, and
- a strengthening of the positive-feedback loop in which cost savings from optimized maintenance schedules are reinvested in more planned maintenance.

Applying these LPs allowed DuPont to save an estimated \$350 million per year by the mid-1990s.

the technologies being used, we have to take on the responsibility for the long-term consequences of the systems we design.^{2,18} 

Acknowledgments

Part of this work has been supported by Refactoring Energy Systems grant EP/R007373/1.

References

1. *The Global Risks Report 2016*, World Economic Forum, 2016.
2. D. Spinellis, "The Social Responsibility of Software Development," *IEEE Software*, vol. 34, no. 2, 2017, pp. 4–6.
3. D.H. Meadows, "Leverage Points: Places to Intervene in a System," Sustainability Inst., 1999.
4. *Transport and Climate Change: Advice to Government from the Commission for Integrated Transport*, UK Commission for Integrated Transport, 2007; <http://www.cambridgeenergy.com/archive/2007-02-08/commission-integ-trans.pdf>.
5. K. Ogata, *System Dynamics*, 4th ed., Pearson, 2013.
6. S. Liu, K.P. Triantis, and S. Sarangi, "A Framework for Evaluating the Dynamic Impacts of a Congestion Pricing Policy for a Transportation Socioeconomic System," *Transportation Research Part A*, vol. 44, no. 8, 2010, pp. 596–608; doi:10.1016/j.tra.2010.04.001.
7. M.G. Vayá and G. Andersson, "Integrating Renewable Energy Forecast Uncertainty in Smart-Charging



BIRGIT PENZENSTADLER is an assistant professor of software engineering at California State University, Long Beach. Her research centers on software engineering for sustainability and resilience; her interests are requirements engineering and infusing sustainability into education. Penzenstadler received a habilitation from the Technical University of Munich's Faculty of Informatics. She's a member of IEEE. Contact her at birgit.penzenstadler@csulb.edu.



STEFANIE BETZ is a professor in Furtwangen University's Department of Computer Science. Her research centers on sustainable software and systems engineering, particularly from the perspective of requirements engineering and business process engineering. Betz received a PhD in applied informatics from the Karlsruhe Institute of Technology. She's a member of ACM. Contact her at besi@hs-furtwangen.de.



LETICIA DUBOC is a researcher in the Department of Engineering at La Salle—University Ramon Llull 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 duboc@salleurl.edu.



NORBERT SEYFF is a professor at the Institute for Interactive 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 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 in the University of Huddersfield's School of Computing and Engineering. 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. He's a member of IEEE and ACM. Contact him at c.venters@hud.ac.uk.



KRZYSZTOF WNUK is an assistant professor in the Blekinge Institute of Technology's Software Engineering Research Group. His research interests include market-driven software development, requirements engineering, software product management, decision making in requirements engineering, large-scale software, system and requirements engineering and management, and empirical research methods. Wnuk received a PhD in software engineering from the Lund University of Technology. Contact him at krzysztof.wnuk@bth.se.



Continued



RUZANNA CHITCHYAN is a senior lecturer in the University of Bristol's Department of Computer Science. Her research centers on requirements engineering and architecture design for software-intensive sociotechnical systems—specifically, applying software engineering techniques to redesign energy systems for sustainability. Chitchyan received a PhD in software engineering from Lancaster University. Contact her at r.chitchyan@bristol.ac.uk.



CHRISTOPH BECKER is an assistant professor at the University of Toronto, where he leads the Digital Curation Institute. 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.



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



- Approaches for Plug-In Electric Vehicles,” *Proc. 2013 IEEE Grenoble PowerTech*, 2013.
8. P.L. Mokhtarian, “Telecommunications and Travel: The Case for Complementarity,” *J. Industrial Ecology*, vol. 6, no. 2, 2002, pp. 43–57.
 9. H. Finidori, “A Pattern Language for Systemic Transformation (PLAST)—(re)Generative of Commons,” presented at 2014 PurplSoc Workshop, 2014.
 10. E. Morozov, *To Save Everything, Click Here: The Folly of Technological Solutionism*, PublicAffairs, 2014.
 11. C.J. Martin, “The Sharing Economy: A Pathway to Sustainability or a Nightmarish Form of Neoliberal Capitalism?,” *Ecological Economics*, Jan. 2016, pp. 149–159; doi:10.1016/j.ecolecon.2015.11.027.
 12. M. Berners-Lee and D. Clark, *The Burning Question: We Can't Burn Half the World's Oil, Coal and Gas. So How Do We Quit?*, Profile Books, 2013.
 13. J.-P. Rodrigue, C. Comtois, and B. Slack, *The Geography of Transport Systems*, Routledge, 2013.
 14. C. Becker et al., “Requirements: The Key to Sustainability,” *IEEE Software*, vol. 33, no. 1, 2016, pp. 56–65.
 15. #SMARTer2030: *ICT Solutions for 21st Century Challenges*, Global e-Sustainability Initiative, 2015.
 16. L. Hilty and B. Aebischer, “ICT for Sustainability: An Emerging Research Field,” *ICT Innovations for Sustainability*, Springer, 2015, pp. 3–36.
 17. P. Checkland, “Soft Systems Methodology: A Thirty Year Retrospective,” *Systems Research and Behavioral Science*, vol. 17, 2000, pp. S11–S58.
 18. C. Becker et al., “Sustainability Design and Software: The Karlskrona Manifesto,” *Proc. IEEE/ACM*

- 37th IEEE Int'l Conf. Software Eng. (ICSE 15), vol. 2, 2015, pp. 467–476.
19. *Oxford English Dictionary*, Oxford Univ. Press, 2012.
 20. United Nations World Commission on Environment and Development, *Our Common Future*, Oxford Univ. Press, 1987.
 21. J.A. Tainter, "Social Complexity and Sustainability," *Ecological Complexity*, vol. 3, no. 2, 2006, pp. 91–103.
 22. K. Proust et al., "Human Health and Climate Change: Leverage Points for Adaptation in Urban Environments," *Int'l J. Environmental Research and Public Health*, vol. 9, no. 6, 2012, pp. 2134–2158.
 23. S. Hodges et al., "Strategies for System of Care Development: Making Change in Complex Systems," Louis de la Parte Florida Mental Health Inst., Univ. South Florida, 2006.
 24. P.C. West et al., "Leverage Points for Improving Global Food Security and the Environment," *Science*, vol. 345, no. 6194, 2014, pp. 325–328.
 25. J.D. Sterman, *Business Dynamics: Systems Thinking and Modeling for a Complex World*, McGraw-Hill Education, 2000.

myCS Read your subscriptions through the myCS publications portal at <http://mycs.computer.org>

Take the CS Library wherever you go!



IEEE Computer Society magazines and Transactions are now available to subscribers in the portable ePub format.

Just download the articles from the IEEE Computer Society Digital Library, and you can read them on any device that supports ePub. For more information, including a list of compatible devices, visit

www.computer.org/epub



IEEE



IEEE computer society