View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Explore Bristol Research

Penzenstadler, B., Duboc, L., Venters, C., Betz, S., Seyff, N., Wnuk, C., ... Becker, C. (2018). Software Engineering for Sustainability: Find the Leverage Points! *IEEE Software*. https://doi.org/10.1109/MS.2018.110154908

Peer reviewed version

Link to published version (if available):
10.1109/MS.2018.110154908

Link to publication record in Explore Bristol Research
PDF-document

University of Bristol - Explore Bristol Research

General rights

This document is made available in accordance with publisher policies. Please cite only the published version using the reference above. Full terms of use are available: http://www.bristol.ac.uk/pure/about/ebr-terms

This article has been accepted for publication in IEEE Software but has not yet been fully edited.
Some content may change prior to final publication.

Software Engineering for Sustainability: Find the Leverage Points!

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 [18].
As software professionals, we can contribute to sustainability through the software systems that we engineer, and it is our social responsibility to do so [21]. But sustainability problems are complex system problems (see 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* – locations within a system where a small change in one aspect can result in significant system-wide changes [10].
This article suggests leverage points (LP) can help software engineers to address sustainability challenges by offering insights on possible transformation mechanisms and/or ways to find alternatives. While LP 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 as they enable intervention on different levels.
We use the example of the UK public transportation system [23] to illustrate how leverage points can contribute to software engineering for sustainability.

SIDEBAR Sustainability

The Oxford English Dictionary [13] defines sustainability as 'the capacity to endure'. The Brundtland commission defined sustainable development as 'meeting the needs of the present without compromising the ability of future generations to meet their needs' [3]. However, to understand the broader sustainability issues, we must ask which system to sustain, for whom, over which time frame, and at what cost [16]. This involves five interrelated dimensions [2]:

- 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.

This article has been accepted for publication in IEEE Software but has not yet been fully edited.

Some content may change prior to final publication.

Running Example: UK Public Transportation System

Existing transport systems are large contributors to greenhouse gas emissions and poor urban air quality, which contribute towards 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 [23]. 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 [14] (see also SIDEBAR System Dynamics).

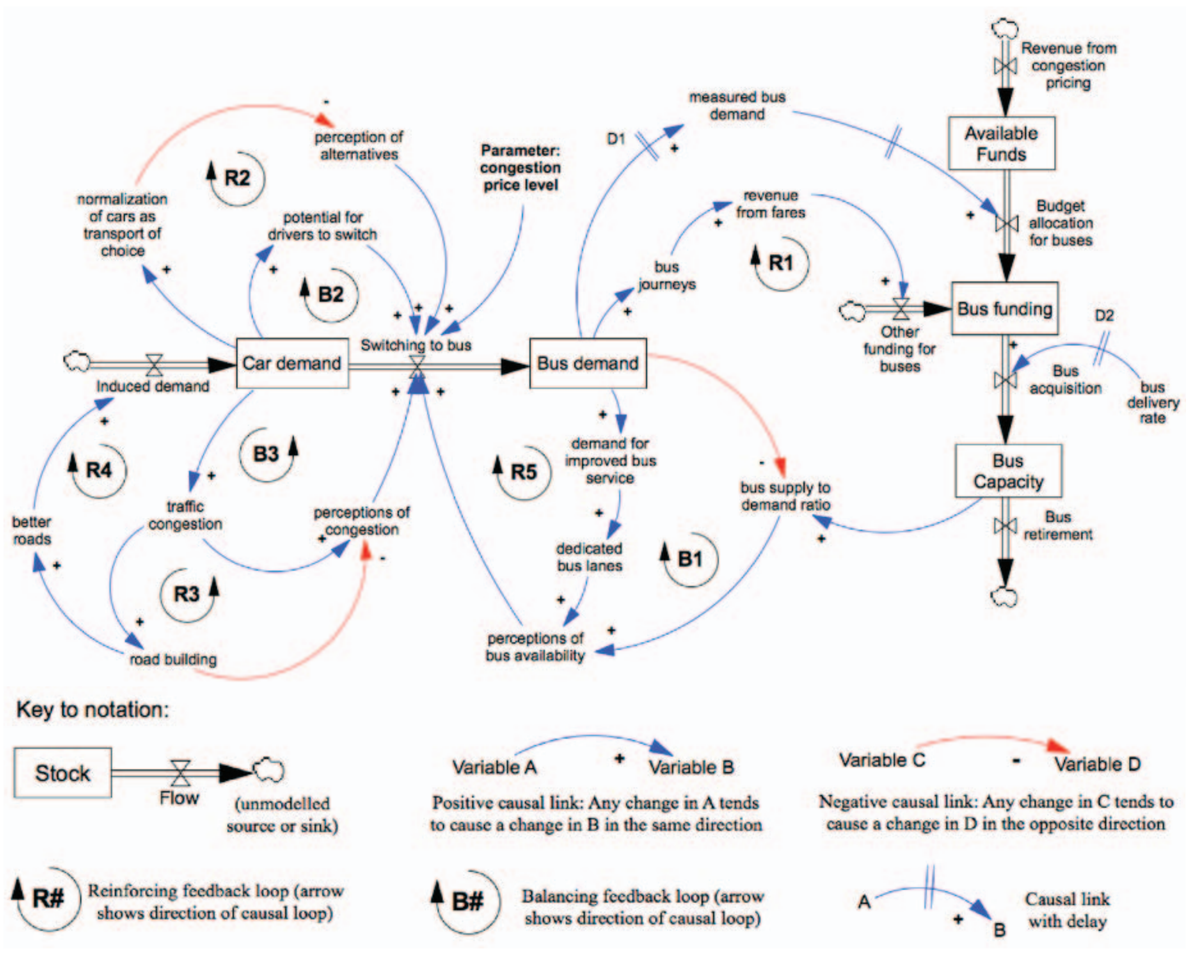

Revenue from congestion pricing
Available Funds
measured bus demand
D1
perception of alternatives
R2
Parameter: congestion price level
normalization of cars as transport of choice
potential for drivers to switch
revenue from fares
R1
Budget allocation for buses
bus journeys
B2
Other funding for buses
Bus funding
D2
Induced demand
Car demand
Switching to bus
Bus demand
Bus acquisition
bus delivery rate
B3
R4
R5
demand for improved bus service
bus supply to demand ratio
Bus Capacity
better roads
traffic congestion
perceptions of congestion
Bus retirement
dedicated bus lanes
B1
R3
road building
perceptions of bus availability
Key to notation:
Stock
Flow
(unmodelled source or sink)
Variable A
Variable B
Positive causal link: Any change in A tends to cause a change in B in the same direction
Variable C
Variable D
Negative causal link: Any change in C tends to cause a change in D in the opposite direction
R#
Reinforcing feedback loop (arrow shows direction of causal loop)
B#
Balancing feedback loop (arrow shows direction of causal loop)
A
B
Causal link with delay

Fig. 1: Causal Loop Diagram of transportation showing contextual factors for mode switching, such as traveller perceptions, funding levels, and relative demand.

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 belief that people need transportation, have some choice over which mode to use, and 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 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

This article has been accepted for publication in IEEE Software but has not yet been fully edited.
Some content may change prior to final publication.

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 transport network is more likely to create a more stable revenue).

The change in stocks due to flows is often non-linear, 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 *B1* in the figure, 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 (*R1*). If this revenue is used to increase the fleet and, consequently, the availability of buses, 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 (*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 (*B3*) [7].

Effects of flows on stocks may not be immediate. For instance, it may take time to gather **information** about changes in demand for public transport (delay *D1* in figure), and even longer to adjust the supply of buses (*D2*) to achieve the **goal** of *increased public transport 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 behaviour. For example, if revenue from congestion charges 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. See [19,20] for further details.

Creating Leverage through Software

How can software effect change in its wider environment? Meadows [10] proposed a list of twelve ways of intervening in a system (*any* type of system), as an invitation to think more broadly about change (see *TABLE Leverage Points)*. We discuss these leverage points in four clusters [5] in increasing order of the likely magnitude of their effect: Changing the metabolic structure, changing the feedback loops, transformational change, and changing the intent of the system and stakeholders.

TABLE Leverage Points

"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" [10]. The leverage points are listed in increasing order of effectiveness according to [10]. While all leverage points *can* bring about change, the later ones are more likely to create significant changes to the system behaviour, but may also require more effort to implement. Meadows' leverage points refer to any kind of change, whether enabled by software or not. In this paper, 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".

LP 12	Constants, parameters, and numbers; tweaking parameters allows change to the intensity of the flows in systems, but rarely alters the underlying dynamics.

This article has been accepted for publication in IEEE Software but has not yet been fully edited.

Some content may change prior to final publication.

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 making it more efficient by optimizing the flow.
LP 10	The structure of material stocks and flows (such as transport 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; 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, 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 mindset or paradigm out of which the system arises; Paradigms are a shared set of deep beliefs about how the world works. They are hardest to change in a system, as society will fiercely resist any challenges to their paradigms.
LP 1	The power to transcend paradigms. This final and most effective leverage point is about being unattached to existing paradigms, there is no certainty in any particular worldview.

Changing the metabolic structure of the system (LP12, LP11, LP10)

These leverage points 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 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, and so often fail to initiate broader change, and sometimes end up entrenching current behaviors [10].

For example, we can view the public transport network of a city as a structure of material stocks and flows with many parameters, including pricing mechanisms, road capacity, bus frequency,

This article has been accepted for publication in IEEE Software but has not yet been fully edited.

Some content may change prior to final publication.

and so on. Some cities use congestion charges, whereby drivers must pay to drive through highly congested areas (top middle Fig. 1.), with higher fees for larger engines that produce more emissions. This can encourage commuters to choose public transit rather than pay 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: Simply increasing the congestion charge within a city will not resolve the congestion problem if commuters believe that they have no other convenient alternatives (*R2* in figure). 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 transport 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. Long-term, data analytics can help to optimize the structure of the transport network, by recommending changes to existing bus routes.

But where 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 (*R3* in figure). Implementing software solutions to tweak pricing strategies, we may divert attention from issues such as 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 [11].

Changing the feedback loops *(LP9, LP8, LP7)*

This cluster of leverage points 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 (*R4* in figure). Such reinforcing loops are powerful, as they amplify change within a system. Meadows [10] points out "A system with an unchecked [reinforcing] loop ultimately will destroy itself." 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 this loop, 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 (*R5* in figure). 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

This article has been accepted for publication in IEEE Software but has not yet been fully edited.

Some content may change prior to final publication.

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 increase in crowding on buses, and the reduction in 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 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 over-compensates because it is applied for too long. For example, in Figure 1, delays in securing funding and procuring new buses (*D2* in figure) can undermine a strategy of getting people to switch, if they cause large swings in the quality of service, so 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.

TABLE Summary of the role of software in our example and the impact per leverage point cluster.

Cluster	**Objective**	**Role of Software**	**Affected element in traffic example**
Changing the metabolic structure (LP12,LP11,LP10)	Optimisation of the bus fleet to balance the number of passengers and buses needed	Monitor/analyse the transport network and road traffic, to understand commuter patterns, the frequency and type of failure that the buses suffer, and then accordingly: a) adjust number of buses to increase flow, b) and calculate size of buffer of spare buses	Buffer: the number of spare buses allows to increase flow (frequency) when necessary
Changing the feedback loops (LP9,LP8,LP7)	Maintain positive perception with respect to bus availability, e.g. reduce perception of overcrowding and bus shortages	Monitors traffic development over time and identifies commuting patterns to reduce overcrowding and bus shortage by a) redeploying buses to particularly busy routes to reduce overcrowding, b) designating additional lanes as “bus only” lanes when traffic congestion increases	Balancing feedback loop: perception with respect to bus availability via a) increase of total number of buses, b) increase in ratio of bus supply to demand
Transformational change / participatory planning (LP6,LP5,LP4)	Altering the structure of information flows for bus schedules and actual departure times to increase convenience of bus use and total demand	a) Inform the users on time when they need to be at the bus stop by providing information on current location of the buses and their impending arrival time via apps, b) instruct maintenance team on preventing frequent failure types	Information flows: send up-to-date bus schedules to app users
Change in the intent of the system and stakeholders (LP3,LP2,LP1)	Public transport 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 as well as personal freedom and increased opportunities.	Ensure that transport meets the whole range of needs by a better support in choosing transportation modes according to calendar sync and traffic; a traffic system that responds to the analysis of data in real time (& includes renewable energy forecasting); supporting for elderly (accessibility); providing info about use of public transport and show how this contributes to environmental health.	Goals for commuting: the whole example is affected as the underlying structure changes

Digital Object Identifier 10.1109/MS.2018.110154908 0740-7459/$26.00 2018 IEEE

This article has been accepted for publication in IEEE Software but has not yet been fully edited.

Some content may change prior to final publication.

Transformational change (LP6, LP5, LP4)

These leverage points encompass transformational changes in systems, namely 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, so 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 leverage points. 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, and they provide a kind of self-adaption, as users can (somewhat) negotiate service-level agreements amongst 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 who you ask [8]: those whose interests were protected by the old system (taxi owners, people who value privacy) regard this as negative, while 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 suggest software engineers can do that in the *TABLE Action Points.* Maybe ride-hailing services (along with self-driving cars) are not radical enough as changes [22]. Because they still emphasize the car as the dominant mode of transport, they lock us into highly energy- and material-intensive forms of personal transport that are ill-suited to dense urban settings, and do little to reduce our environmental footprint [17].

TABLE Action Points for applying LP 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, 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 off 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 the likely results of such intervention be? Has the customer be made aware of it?

This article has been accepted for publication in IEEE Software but has not yet been fully edited.
Some content may change prior to final publication.

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

Change in the intent of systems and stakeholders (LP3, LP2, LP1)

The most radical system changes tend to occur when we change our goals, or the mindset ("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 transport 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 terms goals; what would the sustainable city of the future look like, and what kinds of software are needed to implement them? We need look beyond a client's initial ideas for what they want, and to help them to identify their real goals and the kinds of software capability that would help meet them [2].

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 behaviours 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.

TABLE Evidence for the impact of leverage point analysis

LP analysis has had tremendous impact in many areas, e.g., leveraging feedback loops in development of urban policy in response to climate change in Australia [24]; leveraging a mindset change and goal setting (as well as other LPs) with focus on children's health to improve health care systems in 6 US sites during a 2004-2009 study [25]; and using metabolic structure LPs in global simulations for increasing food security while lowering environmental impact [27]. Furthermore, Sterman documents three major case studies [26]:

General Motors (mid-1990's)	GM analyzed the impact of leasing on new car sales, particularly the rise in availability of high quality off-lease used vehicles. The systems dynamics analysis revealed two key LPs: a policy change towards longer lease terms and a new information flow, to collect data on changes in consumer choice. Other LPs also 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,

This article has been accepted for publication in IEEE Software but has not yet been fully edited.
Some content may change prior to final publication.

	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.
Du Pont (mid-1990s)	After a long history of cost-cutting measures, Du Pont 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 their equipment. Their analysis suggested three key leverage points: 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 "fire-fighting" mentality; and a strengthening of the positive feedback loop in which cost savings from optimized maintenance schedules are re-invested in more planned maintenance. Applying these leverage points allowed DuPont to save an estimated $350million per year by the mid 1990s.

Conclusion

Software is deeply ingrained in our society. Software systems provide, by nature, 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 [6]. 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 leverage points we have access to, and how to deploy them (see evidence in *TABLE Evidence*).

Measuring the impact of applying leverage points 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 [28] 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 leverage points and how to make use of them in software design for sustainability [1].

On a societal level, these leverage points 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 a leverage point. 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 leverage points goes beyond software engineering. It is rather 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 the technologies being used, we have to take on the responsibility for the long-term consequences of the systems we design [1,21].

This article has been accepted for publication in IEEE Software but has not yet been fully edited.

Some content may change prior to final publication.

Take-away: 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 towards sustainability.

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

References

[1] Becker, C., et al. "Sustainability design and software: the karlskrona manifesto." Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on. Vol. 2. IEEE, 2015.
[2] Becker, C. et al. (2016) Requirements: The Key to Sustainability. IEEE Software, 33 (1). pp. 56-65. ISSN 0740-7459
[3] United Nations: World Commission on Environment and Development: Our Common Future. Oxford Univ. Press, 1987
[4] Deek, F., McHugh, J., Eljabiri, O. (2005). Strategic Software Engineering: An Interdisciplinary Approach, Auerbach Publications
[5] Finidori, H. "A Pattern LAnguage for Systemic Transformation (PLAST) - (re)Generative of Commons". PurplSoc Workshop, at Danube-University, November 14/15, 2014
[6] Hilty, L., and Aebischer, B. "ICT for sustainability: An emerging research field." ICT Innovations for Sustainability. Springer International Publishing, 2015. 3-36.
[7] Liu, S., Triantis, K. P., & Sarangi, S. (2010). A framework for evaluating the dynamic impacts of a congestion pricing policy for a transportation socioeconomic system. Transportation Research Part A, 44(8), 596-608. doi:10.1016/j.tra.2010.04.001
[8] Martin, C. J. (2016). The sharing economy: A pathway to sustainability or a nightmarish form of neoliberal capitalism? Ecological Economics, 121, 149–159. doi:10.1016/j.ecolecon.2015.11.027
[9] Meadows, D. H., Meadows, D. L., and Randers, J. (1992). Beyond the Limits: Global Collapse or a Sustainable Future, Earthscan Ltd
[10] Meadows, D. H. (1999) Leverage points - places to intervene in a system. The Sustainability Institute.
[11] Morozov, Evgeny. To save everything, click here: The folly of technological solutionism. PublicAffairs, 2014.
[12] Olson, M. M., & Raffanti, M. A. (2006). Leverage points, paradigms, and grounded action: Intervening in educational systems. World Futures, 62(7), 533-541.].
[13] Oxford English Dictionary. 2012. Oxford Dictionaries
[14] Ogata, K. (2013). System Dynamics, Pearson, 4th edition
[15] Gobal e-Sustainability Initiative: "The Smarter2030 opportunity: ICT Solutions for 21st Century Challenges"
[16] J. A. Tainter "Social Complexity and Sustainability" Ecological Complexity, vol. 3, no. 2, pp. 91-103, 2006
[17] Rodrigue, Jean-Paul, Claude Comtois, and Brian Slack. The geography of transport systems. Routledge, 2013.
[18] World Economic Forum, The Global Risks Report 2016, (11th ed) World Economic Forum, Cologne
[19] Vayá, Marina González, and Goran Andersson. "Integrating renewable energy forecast uncertainty in smart-charging approaches for plug-in electric vehicles." PowerTech, 2013 IEEE Grenoble. IEEE, 2013.
[20] Mokhtarian, Patricia L. "Telecommunications and travel: The case for complementarity." Journal of Industrial Ecology 6.2 (2002): 43-57.
[21] Spinellis, Diomidis. "The Social Responsibility of Software Development." IEEE Software 34.2 (2017): 4-6.
[22] Berners-Lee, Mike, and Duncan Clark. The Burning Question: We can't burn half the world's oil, coal and gas. So how do we quit?. Profile Books, 2013.
[23] UK Commission for Integrated Transport. Transport and Climate Change - Advice to Government. 2007. http://www.cambridgeenergy.com/archive/2007-02-08/commission-integ-trans.pdf
[24] Proust, Katrina, et al. "Human health and climate change: leverage points for adaptation in urban environments." International journal of environmental research and public health 9.6 (2012): 2134-2158.

This article has been accepted for publication in IEEE Software but has not yet been fully edited.

Some content may change prior to final publication.

[25] Hodges, Sharon, et al. "Strategies for system of care development: Making change in complex systems." Tampa, FL: Research and Training Center for Children's Mental Health, Department of Child and Family Studies, Louis de la Parte Florida Mental Health Institute. (2006). http://rtckids.fmhi.usf.edu/cssi/default.cfm

[26] Sterman, John D. Business dynamics: systems thinking and modeling for a complex world. McGraw-Hill Education. 2000.

[27] West, Paul C., et al. "Leverage points for improving global food security and the environment." Science 345.6194 (2014): 325-328.

[28] Checkland, Peter. "Soft systems methodology: a thirty year retrospective." Systems research and behavioral science 17.S1 (2000): S11-58.

BIRGIT PENZENSTADLER is an Assistant Professor at the California State University Long Beach. Her research centers around software engineering for sustainability and resilience. Her interests are requirements engineering and infusing sustainability into education. She received a doctoral degree and a habilitation degree from the Technical University of Munich. She is an IEEE member. Contact her at birgit.penzenstadler@csulb.edu

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

This article has been accepted for publication in IEEE Software but has not yet been fully edited.

Some content may change prior to final publication.

received a PhD in computer science from University College London. Contact her at leticia@ime.uerj.br.

COLIN C. VENTERS is a Senior Lecturer in software systems engineering in the School of Computing and Engineering at the University of Huddersfield. His current research focuses on sustainable software systems engineering from a software architecture perspective for pre-system understanding and post-system maintenance and evolution including architectural recovery, architectural knowledge management and decision making, and architectural-level software metrics and evaluation. Venters received a PhD in computer science from the University of Manchester, UK. He is a member of the IEEE and ACM. Contact him at c.venters@hud.ac.uk.

STEFANIE BETZ is a junior research group leader 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

Digital Object Identifier 10.1109/MS.2018.110154908 0740-7459/$26.00 2018 IEEE

This article has been accepted for publication in IEEE Software but has not yet been fully edited.
Some content may change prior to final publication.

perspective of requirements engineering and business process management. Betz received a PhD in applied informatics from the Karlsruhe Institute of Technology. She is a member of the ACM. Contact her at stefanie.betz@kit.edu.

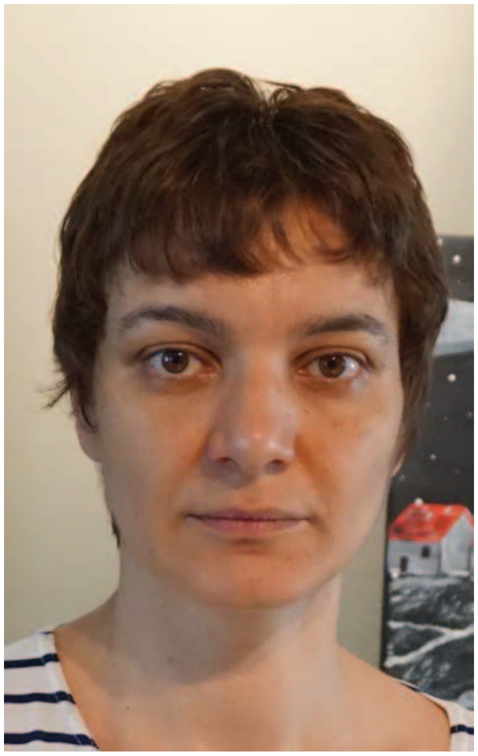
NORBERT SEYFF is a Professor in the School of Engineering and the Institute of 4D Technologies at the University of Applied Sciences and Arts Northwestern Switzerland and a senior research associate in the University of Zurich's Department of Informatics. His research focuses on requirements engineering 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.

KRZYSZTOF WNUK is an Assistant Professor at the Software Engineering Research Group (SERL), Blekinge Institute of Technology, Sweden. 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. He is

This article has been accepted for publication in IEEE Software but has not yet been fully edited.

Some content may change prior to final publication.

interested in software business, open innovation, and open source software. He works as an expert consultant in software engineering for the Swedish software industry. Wnuk received a PhD in software engineering from Lund University of Technology. Contact him at krzysztof.wnuk@bth.se.



RUZANNA CHITCHYAN is a Senior Lecturer at the University of Bristol, Department of Computer Science. Her research centers on requirements engineering and architecture design for software-intensive socio-technical systems - specifically, applying software engineering techniques to re-designing energy systems for sustainability. Chitchyan received a PhD in software engineering from Lancaster University. Contact her at r.chitchyan@bristol.ac.uk.

STEVE M. EASTERBROOK is a Full 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 systems analysis to the challenge posed by global climate

This article has been accepted for publication in IEEE Software but has not yet been fully edited.
Some content may change prior to final publication.

change. Easterbrook received his PhD in computing from Imperial College London. Contact him at sme@cs.toronto.edu.

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.

Tweets (max 115 characters):

- Leverage points challenge us to engage in a (philosophical and practical) discussion of our goals as society
- Leverage points help software engineers see the potential of software for supporting sustainability
- Software engineers can identify leverage points to develop software for transformational change