Sustainability Debt: A Metaphor to Support Sustainability Design Decisions

Stefanie Betz*, Christoph Becker† ‡‡, Ruzanna Chitchyan‡, Leticia Duboc§,
Steve M. Easterbrook†, Birgit Penzenstadler‖, Norbert Seyff**x, Colin C. Venters††
*Karlsruhe Institute of Technology, Karlsruhe, Germany, stefanie.betz@kit.edu
†University of Toronto, Toronto, Canada, {christoph.becker@utoronto.ca; sme@cs.toronto.edu}
‡Department of Computer Science, University of Leicester, UK, rc256@le.ac.uk
§State University of Rio de Janeiro, Brazil, leticia@ime.uerj.br
‖California State University Long Beach, USA, birgit.penzenstadler@csulb.edu
**University of Applied Sciences and Arts Northwestern Switzerland, Windisch, Switzerland, norbert.seyff@fhnw.ch
††University of Huddersfield, Huddersfield, UK, c.venters@hud.ac.uk
‡‡Vienna University of Technology, Austria
xUniversity of Zurich, Switzerland

***Abstract*—Sustainability, the capacity to endure, is fundamental for the societies on our planet. Despite its increasing recognition in software engineering, it remains difficult to assess the delayed systemic effects of decisions taken in requirements engineering and systems design. To support this difficult task, this paper introduces the concept of *sustainability debt*. The metaphor helps in the discovery, documentation, and communication of sustainability issues in requirements engineering. We build on the existing metaphor of technical debt and extend it to four other dimensions of sustainability to help think about sustainability-aware software systems engineering. We highlight the meaning of debt in each dimension and the relationships between those dimensions. Finally, we discuss the use of the metaphor and explore how it can help us to design sustainability-aware software intensive systems.**

***Index Terms*—metaphor; sustainability; debt; design decisions; requirements**

I. Introduction

Software Engineers are beginning to understand the importance of designing sustainability-aware software systems [1], [2], and initial attempts have been made to explore the role of sustainability in software system design (e.g. [3], [4], [5]). Sustainability matters for all software systems, even if the application domain of the system is not related to sustainability, because any new software creates dependencies and obligations as it becomes part of our technical infrastructure, and its on-going use may entail new burdens on social and ecological systems.

Sustainability design is a systemic concept, as it addresses effects that play out in a number of different dimensions, at different levels of a system, over multiple time scales [6]. Because of this, the design process involves trade-offs, in which decisions that improve the perceived success of the software on some level may reduce its overall sustainability in other ways - for example, meeting a release deadline might have an unacceptable impact on the well-being of the development team, or the decision to use a cheaper technology platform might greatly decrease the energy efficiency of a software service. While sustainability is not necessarily in competition with other attributes [3], if these trade-offs are not considered explicitly during the design process, the overall sustainability of the system may suffer. Although software developers are increasingly aware of sustainability as an issue to be addressed, it is currently still challenging to identify and communicate the rationale for sustainability design, especially because of the inherent systemic effects of design decisions.

This paper offers a step forward to support software engineers in sustainability design thinking by introducing a new metaphor: *sustainability debt*. We start from the idea of technical debt, which has proved to be a powerful metaphor for discussing technical and economic trade-offs in software engineering [7]. However, discussion of technical debt has been limited to the trade-offs between current design effort and future software maintainability, which means it focuses primarily on a potential burden for the software development team itself, in the future. By re-framing the metaphor around our broader obligations to society and to future generations, we develop a way of thinking that encourages explicit consideration of these obligations during the design process.

We explore what the idea of sustainability debt means, what it entails, and how this metaphor can be used to identify and communicate about effects of software design decisions on sustainability. Based on that, we identify key questions around how we incur and manage sustainability debt. Finally, we discuss the limitations accompanying the use of a metaphor.

II. Background

A. Sustainability Design

Sustainable development is defined by the UN World Commission on Environment and Development [1987] as *"development that meets the needs of the present without compromising the ability of future generations to meet their own*

needs". In identifying what those needs are, we can examine sustainability with respect to five interdependent dimensions: economic, social, environmental, technical, and human, with the insight that there are both short- and long-term concerns in each dimension [3]. Environmental sustainability is concerned with safeguarding the ecological systems and natural resources that enable life to flourish on planet earth. Social sustainability is concerned with building trust and equity within society. Economic sustainability is concerned with ensuring ongoing prosperity. Individual sustainability is concerned with the freedom, wellbeing and fulfillment of individuals, and finally, technical sustainability is the long-term maintenance and of our technical infrastructures, including software-intensive systems [4].

For software, sustainability includes, for example, the carbon footprint of a software-intensive system during development and use (environmental dimension), how the software changes the ease with which we communicate with one another (social dimension), the level of stress a newly imposed software system imposes on the stakeholders (individual dimension), the maintainability of the software itself (technical dimension), and the health of the business relationship of the vendor and buyer of the software system (economical dimension).

Because of the central role that software-intensive systems now play in society, software engineering carries a crucial responsibility. Consequently, in addition to values like cost, time, safety, reliability, and so on, we must add sustainability as a goal when designing software. Sustainability has been described as "ethics smeared out in space and time" [8]. For software engineers, sustainability design means taking seriously our broader ethical responsibilities, not just to avoid direct harm that might arise from our software, but also to avoid creating systems that may harm future generations, or degrade the environment in which they live. Thus, when practicing sustainability design in software engineering, one must address sustainability explicitly. To do so, we need to raise awareness and help to understand sustainability and its importance in and for software engineering as well as to discuss the sustainability effects and their inter-dependencies on multiple time-scales.

B. Systemic Effects in Software Systems Design

In the context of sustainability, it is very important to explain and understand the different levels of effects software systems can have on sustainability. These can be classified as: direct effects, enabling effects, and structural effects [9], [10]:

- Direct (or 'first order') effects are concerned with the immediate impacts resulting from the production, use and disposal of software systems. This can be measured using metrics based on performance requirements or network bandwidth for example. The direct environmental impact is quite often measured using the Life Cycle Assessment (LCA) [11].
- Enabling (or 'second order') effects are concerned with the benefits and impacts of ongoing use of the software system. This might be for example how a web search engine reduces the cost of access to information.
- Structural (or 'third order') effects are concerned with changes resulting from the use of software systems by a very large number of people over medium to long term, leading to substantial changes in societal structures such as new laws, politics, or social norms, or economic structures such as the networked economy.

Any system will exhibit such effects over multiple timescales in each of the dimensions [3]. Thus, when designing a system we need to take sustainability dimensions and effects into account. These different dimensions and effects over multiple timescales are interdependent. Changes in one dimension frequently lead to effects over multiple timescales in the different dimensions. Some of these effects are hidden and may show themselves only on the long run. For example, we might think were doing fine in one dimension - economic, say, as we launch a software company with a sound business plan that can make a steady profit - but often we do so only by incurring a debt in other dimensions, perhaps harming the environment by contributing to the mountains of e-waste, or harming social sustainability by replacing skilled jobs with subsistence labour. Thus, it is important to make sustainability effects visible, and to be able to decide what is the most effective way to make a system sustainable. For example, we need to compare alternative actions and consider opportunity costs in sustainability design.

However, when taking decisions in the process of requirements analysis and systems design, it is often very difficult to understand the effects of these decisions. The systemic effects are often remote and dispersed and manifest only over time. The lack of visibility of such effects makes it very difficult to identify the hidden impact of system design decisions in the requirements process and in systems design. Yet, these effects are very real, and identifying them starts with understanding the requirements for the system: For example, the inclusion of a particular stakeholder in the elicitation process may have far-reaching consequences about the goals and objectives that are identified, the types of scenarios that will be explored, and the success criteria that will be specified. The systemic nature of sustainability makes it challenging to identify possible risks, trade-offs and effects, and hinders effective communication across diverse stakeholders. This makes it difficult to discover, document, and communicate issues of sustainability as part of RE and throughout systems design.

C. The Metaphor Technical Debt

Debt is usually thought of as a financial tool to facilitate economic transactions over time: For example, it enables prospective home owners to buy property without possessing the full funds, at a certain cost (the interest).

A more detailed definition is given by Ampatzoglu et al. [7], who provide a glossary of terms for technical debt grounded in financial terminology. In this more formal view, debt is *"used to describe the amount of money owed by one party (**debtor** or borrower) to another party (**creditor** or lender). The certain*

amount of money derives from a loan, which denotes that the money has been lent by the creditor to the debtor for a specific period of time. The obligation of the debtor is to repay a larger sum of money to the creditor at the end of that ***period*** *[28]. The original amount of money borrowed is called the* ***principal****, while the additional amount paid back constitutes the* ***interest****."* [12] Hence, financial debt implies at least the recognized existence of a debtor, a creditor, and a contractual obligation of the debtor to the creditor that is more or less quantified, and (usually) an identified period of time.

This set of financial models and relationships grounds the metaphorical use of debt in clearly defined terms and points to a range of typical activities and transactions that arise when handling debt. We can attempt to apply financial and economic models such as using interest rates to quantify the future value of a current action and furthermore, to identify, measure, prioritize, repay, and monitor debt [13]. Finally, we can distinguish between intentional actions of taking on debt for strategic reasons and unintentional consequences of careless or reckless decisions [14].

To help think about the technical sustainability of software systems and consider opportunity costs, the metaphor of technical debt has been introduced [15]. Originally coined in an experience report, it is based on the idea that software engineers sometimes intentionally or unintentionally generate "debt" in one area (e.g. software quality) to meet the needs of another area (e.g. a deadline) and underestimate the long-term consequences of their decisions. Metaphors are about "understanding and experiencing one kind of thing in terms of another." [16]. They structure something that is less clearly understood (such as time) in terms of another kind of thing of which we have a clearer grasp (such as money). Contrary to common thought, metaphors are fundamental cognitive tools that we use to form the conceptual models we use for understanding the world [16]. The technical debt metaphor helps software engineers to understand these trade-offs and communicate about them. Furthermore, it helps us to identify temporal effects of designing and developing a system the cheap and easy way, without thinking about consequences, by thinking about a technical "loan" we take on and the "interest" we will pay depending on the amount of that loan and the conditions we will find ourselves in. Being able to quantify these costs enables us, in theory, to take more conscious, well-informed decisions based on a cost-benefit trade-off.

Technical debt has been applied and elaborated over two decades, and a considerable amount of theory and empirical evidence has been accumulated that stands testimony to its usefulness as a metaphor: see, for example, the IEEE Software Special Issue on Technical Debt [7] and the ongoing workshop series on Managing Technical Debt (MTD) starting from 2010 [17]. Nevertheless, there is still a lack of theoretical models, methods and practical tools to identify, measure, manage, and reason about technical debt in software engineering [7].

III. Sustainability debt

A. Introducing Sustainability Debt

We propose the concept of sustainability debt as a generalization of technical debt, and as a metaphor for thinking about negative effects of software systems in the five sustainability dimensions. This metaphor re-casts what to many people is a very difficult and abstract concept (sustainability and the delayed effects of our design decisions) in terms of a concept we have a clearer conceptual structure for (debt). This enables us to bring multiple strands of thoughts together, and explore the implications for concrete requirements activities conducted as part of software and systems engineering.

To extend Ward Cunningham's concept of "technical debt" we offer the following **working definition**, inspired by Kruchten [7] and Hilty [18]:

"Sustainability debt is the hidden effect of past decisions about software-intensive systems that negatively affect economic, technical, environmental, social, and individual sustainability of the system under design. Effects in these dimensions can manifest themselves on three different levels: (1) the direct effects of the software system production and use; (2) enabling effects that arise from the ongoing use of the software system, and (3) systemic changes caused by the use of the software system on a larger scale over time."

B. The Five Sustainability Debts

Considering the multi-faceted nature of sustainability, we need to take into account inter-dependencies between the different sustainability dimensions. A change in one dimension might have a negative hidden impact in other dimensions. At the same time, we need to be careful not to think that we can exchange and transfer value between dimensions at will; many of these issues are not commensurate and cannot be compared directly.

We therefore argue it is important to assess sustainability debt separately in each of the five dimensions:

- Economic Sustainability Debt is the hidden effect of decisions about a software system that negatively affect time and cost. It is closely based on the well-known concept of financial debt.
- Technical Sustainability Debt is the hidden effect of decisions about software that negatively affect the software system itself, such as the understandability and maintainability of the code.
- Environmental Sustainability Debt is the hidden ecological burden that arises from the lifecycle of a software system from its creation to its disposal. This includes the demand for resources and energy and the generation of pollution and wastes that arise from the systemic effects of its ongoing use.
- Social Sustainability Debt is the hidden effect of decisions about software that negatively affect social justice, equity, and fairness, or which lead to an erosion of trust in society.

- Individual Sustainability Debt is the hidden effect of past decisions about software that negatively affects individual freedom and fulfillment by imposing constraints and restrictions on individual stakeholders.

The relationships between these dimensions and how they affect the overall sustainability debt is complex - influences might be reinforcing or canceling each other out. Also, incurring a debt in one dimension can have an effect on any of the other dimensions. We know that we cannot list and explain them in full detail, even more so as we cannot know all of them. Nevertheless, it is important to mention that sustainability debt is more than just the sum of the five dimensions debts because of the inter-dependencies between them.

We are neither claiming that is possible to measure one overall sustainability debt nor that it is possible to express the debt in each dimension in monetary terms. Many of the factors in the dimensions other than financial represent externalities in the economic system, and they are treated as externalities because its remarkably hard to agree on how to price them: How do you put a value on a human life? On a walk in the forest? On freedom?

Thus, we seek appropriate measurable indicators of the level of debt within each dimension, but we do not attempt to seek a direct translation of the measures between dimensions. Instead, we suggest to assess the trade-offs qualitatively, through a participatory approach including stakeholders.

We next describe each of these debts in detail.

1) Economic Sustainability Debt: Economic debt is directly based on financial debt - the hidden effect of past decisions about software that negatively affects time, costs and other economic issues. As such, it primarily focuses on assessing financial aspects such as software development costs, and many of these are already at the focus of decisions in software engineering [19].

However, we consider economic debt to go beyond basic financial aspects and to also look at broader economic aspects and values.

Key concepts in this area include Net Present Value (NPV), an attempt to measure the economic value of a software product, and the Total Cost of Ownership (TCO) of a software system. Other models help to measure trade-offs and make decisions about future developments: This includes Real Options Analysis, opportunity costs and other financial mechanisms. Finally, Value-Based Software Engineering [20] is an attempt to broaden the economic perspective of values in SE.

Virtually all decisions taken during software development have an impact on the economic dimension of sustainability in that they require effort and resources. The following example highlights specific situations that we can conceive of as intentionally taking on economic debt:

- In order to follow the idea of sustainability design, the designers of a supply chain system may want to involve a larger group of stakeholders within requirements elicitation, including community members, service delivery companies and suppliers. They may also want to apply a wider range of techniques that help to identify goals and risks in all dimensions of sustainability, such as a reference sustainability goal model [4]. This will increase the immediate costs of requirements analysis for the intended benefit of a system that is perceived as more valuable. Other decisions will have less visible economic effects. Identifying and uncovering these can facilitate a more explicit documentation and conversation in the systems design process.

2) Technical Sustainability Debt: Cunningham [15] [21] is attributed with coining the term technical debt as a metaphor for explaining the value of refactoring to non-technical stakeholders: “Shipping first time code is like going into debt. A little debt speeds development as long as it is paid back promptly with a rewrite...The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented otherwise.“ [15]. Tate [22] defines technical debt as an accumulation of poor design decisions over time which result in an inability to develop new features due to the defect burden incurred. McConnell [23] extends this definition by proposing a technical debt taxonomy, which distinguishes between *unintentional* (Type I) and *intentional* (Type II) debt where the former is the result of poor engineering practice, and the latter is a conscious, strategic decision to optimize for the present rather than for the future. He also distinguishes between *short-term* (Type II A) and *long-term* (Type II B) debt where the former is taken on *tactically* and *reactively* but is expected to be paid off frequently, and the latter is taken on *strategically* and *proactively* and paid off sometime in the indeterminate future. He suggests that when technical debt is incurred for strategic reasons, the fundamental reason is always financial pressures. Lavalle and Robillard [24] highlight other issues that can lead to debt. They investigated how organizational factors such as structure and culture have an impact on the working conditions of developers. Their observations revealed that many decisions made under the pressure of certain organizational factors and identified ten issues that had a negative impact on software quality including documenting complexity, internal dependencies, external dependencies, human resource planning, organizational politics, undue pressure from managers, scope protection, budget protection, organically grown processes, and lack of vulnerability testing. They conclude that while none of these issues are new, any software development project should present few of these issues and ideally none of them.

McConnell [23] also introduces the concept of non-debt where examples may include a features backlog, deferred features, cut features, etc. suggesting that not all incomplete work is debt in itself as a result of it not accruing interest payments. This is a position supported by Fowler [14] who suggests that it is not useful to distinguish between debt or non-debt, but between *prudent* versus *reckless* debt, and *deliberate* versus

inadvertent debt i.e. technical debt quadrant. Kruchten, Nord and Ozkaya [7] extend this stating that technical debt should be confined to describing the invisible result of past decisions about software that negatively affect its future.

Fowler [25] argues that the principal problem with technical debt is the lack of quantification. To address this Kazman et. al., [26] argue that architectural debt, hotspots within that architecture, are the principal causes of technical debt; architectural roots, a type of technical debt that incurs high maintenance penalties. Based on a case study they identified and quantified the architecture debts in a large-scale industrial software project, and justified the refactoring of architecture problems with an economic analysis which suggested that in financial terms, the project could expect a 295 % return on investment in the first year alone. This is a position supported by Ozkaya [27] who suggest that an architecture-focused analysis approach helps manage technical debt by enabling software engineers to decide the best time to pay down the technical debt.

Examples of technical debt are:

- When engineers choose to use technical workarounds, to skip validation, or to rush testing to deliver the product in time, they are causing architecture instability.
- Another example is a system where the energy efficiency of algorithms is improved, but this might be making the code more difficult to understand and modify.

3) Environmental Sustainability Debt: Guzman, Piattini and Prez-Castillo [28] proposed the concept of ecological debt as "the cost (in terms of resource usage) of delivering a software system with a greenability degree under the level of the nonfunctional requirements established by stakeholders, plus the incurring cost required to refactor the system in the future". This debt can be quantified as follows:

$EcologicalDebt = \sum Cost(Resource_i) + \sum Refactor(EcologicalFlaw_j)$

It is important to note that, ecological debt is defined in terms of the *greenability* of the system, which the authors refer to as "a software quality characteristic for measuring the degree to which a software product has appropriate power consumption". Furthermore, although referring to the same dimension, ecological debt still differs from the concept of environmental debt, proposed in this paper. The latter refers to a wider range of environemental impact that goes beyond resource and energy use, such as pollution and imbalance of the ecosystem caused by the software-intensive system overtime.

We define environmental sustainability debt as the impact of past decisions about software that negatively affects the natural environment. According to Schwarz et al. [29], some of these negative impacts are energy-intensity, expressed as a measure of the net energy consumed to provide the power requirements for the system, and material intensity expressed as of material wasted [29]. The 5 metrics they propose are material intensity, energy intensity, water consumption, toxic emissions, and pollutant emissions. Energy-intensity can be used to track performance of processes for which net energy consumed is negative. The ability to measure progress over time is critical. Schwarz et al.s' argument for providing few measures is that measurements that combine too many components are less versatile and less useful for making comparisons across products and industries. However, the reduction to energy-intensity and material intensity only accounts for direct effects and neglects indirect effects. We argue to also include the (sometimes invisible) aspects of natural resources consumed indirectly, namely greenhouse gas emissions, pollution, and waste production of any kind caused directly or indirectly. The direct effects of energy and resource consumption are pretty straightforward. As soon as we switch on a piece of hardware than runs software, energy is consumed. For indirect effects, this might be less obvious, but here are examples where software systems can cause environmental harm:

- An agricultural system (or oil production system) might have a software system that manages the collection of data from sensors over time. If that data is stored properly, it can be analyzed later on to understand systemic effects on soil; if not, it will have an indirect negative effect on the environment as the effects on the soil caused by that system will not be documented.
- A supply chain management system is part of a production or manufacturing system. There can be a design choice to show the environmental aspects of the production parts or processes to choose from with respective metrics of material and energy intensity they require, such that the manager can make more informed choices.
- A similar sort of systems is used in oil production in order to collect data from different depth levels of the sea to analyse the impact of drilling and extracting oil. Data for the system is gathered using a trawler net that collects specimen of the fauna to analyse them potentially causing harm to their catch but hopefully for keeping the overall population safe.

These three examples show that environmental debt can occur by direct and indirect effects of software systems and should be made more visible, accessible, and traceable.

4) Individual Sustainability Debt: Individual sustainability is concerned with the overall well-being of an individual. From a social perspective, maintaining an environment that supports individual sustainability requires investment in health, education, skills, knowledge, participation and leadership, and access to services for the individuals within a given community.

Individual sustainability debt not only results in diminished contributions that an individual would provide to the society and economy of ones placement, but it is also an important source of accumulating social debt. Historically, the notion of "investment into human capital" has been justified by the human "return on the investment" through increased productivity, commitment, and creativity. We prefer to frame the notion of *individual sustainability debt* as the effects of design decisions that reduce the individuals' abilities to freely develop to their fullest potential. This includes impingement of an individuals freedom or privacy, reduced assess to resources

and services, and negative effects on one's own sense of value and fulfillment. For example, intrusions into the private sphere of software users reduces their level of privacy and thus incur a debt on the individual dimension. Unfortunately, this debt is often externalized so that the users carry the effects, and remains invisible for the most part.

As all other kinds of sustainability debt, this one too, is often driven by financial gain, or resistance to change:

- The more customizable a software is, the more development and testing it requires, so it is cheaper and faster to develop a one solution that fits all users. A typical way of reasoning would be: This software is not for the visually impaired, so we do not need to consider text to voice.
- If we do not allow users to opt out of our information collection plan for the sake of their privacy, we can sell this data to others and generate more advertisement revenues.
- If we allow flexible working arrangements, we need to set up a system to monitor, support, and evaluate these arrangements. It is easier to have all work from the office from 9.00 till 17.00.

5) Social Sustainability Debt: One of the first times "social debt" has been mentioned was in the sociological science in the 60s by Muir and Weinstein [30]. They claimed that humans (especially from the western world) tend to transfer the norms of economic behavior using creditor and debtor roles into the social dimension and that financial debt is accompanied with strained social relationships between creditor and debtor. Thus, they defined social debt as strained social relationships imposed on stakeholders that emerge as a consequence of inequality [debtor-creditor] circumstances [30]. In software engineering the term social debt has first been coined in by Tamburri et al. in 2013 [31]. They did not provide an exact definition regarding it but rather explained it as unforeseen project costs connected to strained social relationships caused by for example uninformed socio-technical decisions. Additionally, they stated that social debt is connected to technical debt and that it is intertwined with technical debt.

Nevertheless, for us this understanding is too narrow. It only covers one part of social debt in the context of software engineering respectively sustainability design: It focus only on development communities. In the context of sustainability debt, we extend the focus on the social communities affected by and related to the system under design. This can be as wide as society as a whole. Thus, social debt is sustainability debt in the dimension of social justice, equity, fairness for society. Moreover, we argue to not only include the direct effects of social debt but also indirect effects of social debt. In the following we describe the concept of social debt using examples:

- Consider a supply chain system that changes the control structure from human-based control and direct communication to a mediated form, where the algorithms are in charge and direct communication channels are discarded in the sake of efficiency and traceability. This would undermine social relations with suppliers.
- A globally distributed development project covering different time zones may for example cause a communication delay between stakeholders of the development community leading to a development delay (direct effect) but also to a lack of trust between stakeholders of the development community caused by the distance between them (indirect effects).

IV. Discussion

Sustainability debt considers effects not only on software systems, its development community, and its evolution process, but includes all stakeholders involved and affected.

We do not seek to monetize sustainability debt providing one financial figure. Rather, we introduce sustainability debt as an extension of the technical debt metaphor to provide a basis for discussion and assessment [32] of hidden effects of decisions about software-intensive systems that negatively affects economic, technical, environmental, social, and individual sustainability.

Requirements Engineering plays an important role in that, it is there that we frame our decisions that have far-reaching effects. For example, considering only time and budget as the projects success criteria leads to environmental and social damage (environmental debt). On the other hand, we can incur economic debt in investing efforts to use a wider, participatory set of techniques to identify economic goals and the effects of the system in the environmental dimension.

In Requirements Engineering, we identify elements of sustainability debt and highlight possible effects and debts. The metaphor supports us in identifying and documenting these effects and facilitates communication across a broader set of stakeholders:

- Identify: Five dimensions make the abstract concept of sustainability more tangible and facilitate asking the question "Who is the creditor?"
- Communicate: Debt is well understood, adding the five dimensions makes it tangible across a wide range of stakeholders.
- Document: Ways of documenting and visualization can be developed to facilitate understanding.

The metaphor of sustainability debt encourages requirements engineers to assess the effects on the sustainability of the system under design, and it raises questions such as:

- What can requirements engineers do to identify *causes* of sustainability debt? The causes of sustainability debt are not necessarily different from the causes of technical debt such as schedule and budget constraints, lack of vision, plan strategy, unclear requirements, bad assumptions etc. [33]. However, these are very high-level causes. Can we provide more specific causes and understand the underlying perceptions and assumptions behind the decisions that cause debt? How is reckless sustainability debt different from other situations?
- Debt implies something quantifiable. Can sustainability debt be *quantified* in requirements engineering? We argue

it is not reducible to a single indicator. But, we argue that should be possible to develop meaningful indicators for the different dimensions, and that we need to develop mechanisms to qualitatively assess trade-offs.
- This leads directly to the discussion to intentionally take a debt as *strategic decision*, for example to shorten time to market. Can sustainability debt be used in a similar intentional way? For example, in the dimension of individual debt, this can simply refer to imposing additional stress on developers to speed up development time. But does this also work for the overall sustainability debt? Do we want that?

These questions are pointing to the limitations of a metaphor highlighting some effects and masking others: A debt usually implies a quantifiable contract between debtor and creditor. In sustainability debt, this is not necessarily the case: We are only able to quantify the debt to some extent (if at all), and the creditor is quite often quite generic and removed in space and time – in the extrem case, future society as a whole – or only represented by surrogate stakeholders. Hence, no mechanism of repaying debt may be known when incurring the debt. This raises the question whether sustainability debt is the right metaphor to use at all. Consider that some debt can never be paid back, such as irreversible environmental degradation, or that the consequences of debt in one dimension can result in irreversible debt in a different dimension, e.g. environmental debt can lead to individual losses in health.

Despite these limitations, it is important to highlight that it is only based on this metaphorical thinking that we are enabled to raise these particular questions. By emphasizing the notion of incurring debt, these invisible effects of decisions are made visible and the notions of interest and repayment surface. Further consequences of the concept have yet to be explored.

V. Related Work

We were not the first to use the metaphor of debt to discuss how past decisions about software-intensive systems may have a negative effect on sustainability. However, we are the first ones defining and discussing sustainability debt for the different dimensions, including the different levels of effects and their interrelations.

Ojameruaye and Bahsoon also use in their white papers [34], [35] the concept of sustainability debt to describe another form of technical debt, which provides a metric and quantifies the gap between the level of sustainability that will be achieved with a specific architecture and an ideal environment where the sustainability requirements are completely achieved. To deal with this debt, the authors propose an economic-driven architectural evaluation method that helps identifying decisions that minimize costs and risks, while maximizing value on the five sustainability dimensions. In their method, the impact of alternative architectural strategies on sustainability is determined using a set of value indicators along four perspectives [36]: financial, customer, internal process, innovation and learning. These perspectives are mapped to the sustainability dimensions using indicators. The indicators include maintainability, total development and implementation costs, and perceived value, among others. Some of the indicators are taken from [36], while others have been added, such as environmental impact due to energy usage and CO_2 emissions. However, it remains unclear whether additional indicators are needed to measure sustainability in each of the different dimensions.

Their approach is quantitative: they assign integer values from 1 to 5 to represent the impact of architecture parameters (e.g., location finding technology, connectivity technology and database) on the selected value indicators. The same value scale is applied to quantify the risk of each impact, based on criticality and likelihood. These values are subsequently used to rank the candidate architectures, through CBAM [37] and portfolio theory [38]. Finally, the debt is calculated as the gap between the ideal architecture where all goals are achieved and what can be achieved within the given context for the different ranked architectures.

While using the same metaphor, Bendra and Bahsoons aim and understanding of sustainability debt are different and narrower than ours. Their aim is to develop a decision-support framework for choosing the best software architecture with respect to sustainability. The concept they introduce as sustainability debt is an extension of the technical debt with different dimensions. They do not attempt to provide a clear definition for the different types of debts. They also argue that this debt is quantifiable as the gap between the ideal architecture and what actually can be achieved given the available resources. Finally, they do not consider the different levels of effects into the sustainability dimensions as part of the debt.

Our work, in contrast, defines the concept of debt for each sustainability dimension, taking into account the different orders of effect. Technical debt is therefore related to one of the dimensions of the sustainability debt. Most importantly, our aim is a more philosophical one: to define and discuss how the metaphor can be used in each dimension of sustainability and, by doing so, to provide a mental framework for software engineers to think about sustainability during the software development activities. Additionally, we do not claim that sustainability debt can be quantified. Moreover, we think it is arguable to express the debt in each dimension in financial terms or to calculate one overall sustainability debt. Instead we provide a theoretical framework to uncover, represent, communicate, and enable sustainability within software engineering.

VI. Conclusion

We presented the metaphor of sustainability debt as a mental tool to uncover, represent and communicate issues related to sustainability within software engineering. Sustainability debt is the hidden impact of past decisions about software-intensive systems that negatively affects economic, technical, environmental, social and individual sustainability on the systems under design. We foresee that using the sustainability debt metaphor in sustainability design decisions can support practitioners in making the abstract concept of sustainability

more tangible. This can help them to better envision the systemic effects of their design decisions. We have started to discuss the relationships between the presented sustainability debt dimensions here as a first step towards elaborating and measuring sustainability debt.

We envision a taxonomy of sustainability debt as an important step in elaborating on the definition and the structure of the metaphor. Another course of action in the area of measurement and prioritization is to identify and develop appropriate indicators to measure the debt of each dimension such as the Composite Sustainable Development Index [39] or the Schwartz's metrics [29] for environmental debt. Following this, a mechanism needs to be developed and established to enable a joint assessment of trade-offs including relevant stakeholders. Additionally, we propose to investigate the usage of simulation models such as system dynamics to support the understanding of systemic effects. Currently we are working on visualization and communication of the systemic effects of design decisions.

ACKNOWLEDGMENTS

This work is supported by the DFG EnviroSiSE project under grant number PE2044/1-1, by FAPERJ (APQ1), by CNPQ (No 14/2014), by NSERC (RGPIN-2014-06638) by WWTF through project BenchmarkDP (ICT2012-46), by the European Social Fund, and by the Ministry of Science, Research and the Arts Baden-Wuerttemberg.

REFERENCES

[1] B. Penzenstadler, V. Bauer, C. Calero, and X. Franch, "Sustainability in software engineering: a systematic literature review," *16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)*, pp. 32–41, 2012. [Online]. Available: http://digital-library.theiet.org/content/conferences/10.1049/ic.2012.0004

[2] J. Mankoff, R. Kravets, and E. Blevis, "Some computer science issues in creating a sustainable world," *Computer*, vol. 41, no. 8, pp. 102–105, Aug 2008.

[3] C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, M. Mahaux, B. Penzenstadler, G. Rodríguez-Navas, C. Salinesi, N. Seyff, C. C. Venters, C. Calero, S. A. Koçak, and S. Betz, "The Karlskrona manifesto for sustainability design," *CoRR*, vol. abs/1410.6968, 2014. [Online]. Available: http://arxiv.org/abs/1410.6968

[4] B. Penzenstadler and H. Femmer, "A generic model for sustainability with process-and product-specific instances," in *Proceedings of the 2013 workshop on Green in/by software engineering*. ACM, 2013, pp. 3–8.

[5] S. M. Easterbrook, "From computational thinking to systems thinking: A conceptual toolkit for sustainability computing," in *ICT for Sustainability 2014 (ICT4S-14), Stockholm, Sweden, August 25, 2014.*, 2014. [Online]. Available: http://dx.doi.org/10.2991/ict4s-14.2014.28

[6] C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, B. Penzenstadler, N. Seyff, and C. C. Venters, "Sustainability Design and Software: The Karlskrona Manifesto," in *Proc. of the Int. Conf. on Software Engineering*, 2015.

[7] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *IEEE Software*, vol. 29, no. 6, pp. 18–21, November 2012.

[8] J. Garvey, *The ethics of climate change: right and wrong in a warming world*. Continuum International Publishing, 2008. [Online]. Available: http://books.google.ca/books?id=Ng5Nn3Kn3xwC

[9] F. Berkhout and J. Hertin, "Impacts of information and communication technologies on environmental sustainability: Speculations and evidence," *Report to the OECD, Brighton*, vol. 21, 2001.

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

[11] U. E. P. Agency, "Defining Life cycle Assessment," 2010. [Online]. Available: http://www.gdrc.org/uem/lca/lca-define.html

[12] A. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "The financial aspect of managing technical debt: A systematic literature review," *Information and Software Technology*, vol. 64, pp. 52–73, 2015.

[13] Z. Li, P. Liang, and P. Avgeriou, "Architectural debt management in value-oriented architecting," *Economics-Driven Software Architecture, Elsevier*, pp. 183–204, 2014.

[14] M. Fowler, "Technical debt quadrant," http://martinfowler.com/bliki/TechnicalDebtQuadrant.html. [Online]. Available: http://martinfowler.com/bliki/TechnicalDebtQuadrant.html

[15] W. Cunningham, "The wycash portfolio management system," in *OOPSLA'92: Proceedings of Seventh Annual Conferenceon Object-Oriented Programming Systems, Languages, and Applications*, 1992.

[16] G. Lakoff and J. M., "Metaphors we live by," *Chicago/London*, 1980.

[17] S. CMU, "Technical debt workshop series," http://www.sei.cmu.edu/community/td2012/previous/?location=secondary-nav&source=655951. [Online]. Available: http://www.sei.cmu.edu/community/td2012/previous/?location=secondary-nav&source=655951

[18] L. M. Hilty and B. Aebischer, "Ict for sustainability: An emerging research field," in *ICT Innovations for Sustainability*, L. M. Hilty and B. Aebischer, Eds. springer., 2015.

[19] B. Boehm, *Software Engineering Economics*. Prentice-Hall, 1981.

[20] S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Grünbacher, *Value-based software engineering*. Springer Science & Business Media, 2006.

[21] W. Cunningham, "Debt metaphor," https://www.youtube.com/watch?v=pqeJFYwnkjE. [Online]. Available: https://www.youtube.com/watch?v=pqeJFYwnkjE

[22] K. Tate, *Sustainable Software Development: An Agile Perspective*. Addison Wesley, 2006.

[23] S. McConnell, "Technical debt," http://www.construx.com/10x_Software_Development/Technical_Debt/. [Online]. Available: http://www.construx.com/10x_Software_Development/Technical_Debt/

[24] M. Lavallee and P. N. Robillard, "Why good developers write bad code: An observational case study of the impacts of organizational factors on software quality," in *ICSE'15: Proceedings of the International Conference on Software Engineering*, 2015.

[25] M. Fowler, "Technical debt," http://martinfowler.com/bliki/TechnicalDebt.html. [Online]. Available: http://martinfowler.com/bliki/TechnicalDebt.html

[26] R. Kazman, Y. Cai, R. Mo, Q. Feng, L. Xiao, S. Haziyev, V. Fedak, and A. Shapochka, "A case study in locating the architectural roots of technical debt," in *Proceedings of 37th IEEE International Conference on Software Engineering (ICSE15)*, 2015.

[27] I. Ozkaya, "Developing an architecture-focused measurement framework for managing technical debt," 2015-06-15.

[28] I.-R. de Guzmn, M. Piattini, and R. Prez-Castillo, "Green software maintenance," in *Green in Software Engineering*, C. Calero and M. Piattini, Eds. Springer International Publishing, 2015, pp. 205–229. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-08581-4_9

[29] J. Schwarz, B. Beloff, and E. Beaver, "Use sustainability metrics to guide decision-making," *Chemical Engineering Progress*, vol. 98, no. 7, pp. 58–63, 2002.

[30] D. E. Muir and E. A. Weinstein, "The social debt: An investigation of lower-class and middle-class norms of social obligation," *American Sociological Review*, pp. 532–539, 1962.

[31] D. Tamburri, P. Kruchten, P. Lago, H. Van Vliet *et al.*, "What is social debt in software engineering?" in *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on*. IEEE, 2013, pp. 93–96.

[32] I. Ozkaya, P. Kruchten, R. L. Nord, and N. Brown, "Managing technical debt in software development: Report on the 2nd international workshop on managing technical debt, held at icse 2011," *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 5, pp. 33–35, Sep. 2011. [Online]. Available: http://doi.acm.org/10.1145/2020976.2020979

[33] E. Lim, N. Taksande, and C. Seaman, "A balancing act: What software practitioners have to say about technical debt," *Software, IEEE*, vol. 29, no. 6, pp. 22–27, Nov 2012.

[34] B. Ojameruaye and R. Bahsoon, "A portfolio-based approach for evaluating sustainability requirements and their debts in architectures," School of Computer Science, University of Birmingham, UK, Tech. Rep. 2, 2015.

[35] B. Ojameruaye and R. Bahsoon, "Sustainability debt: An economics driven approach for using technical debt analysis in decision making

for sustainable requirements," School of Computer Science, University of Birmingham, UK, Tech. Rep. 3, 2015.

[36] M. Khurum, T. Gorschek, and M. Wilson, "The software value map - an exhaustive collection of value aspects for the development of software intensive products," *Journal of Software: Evolution and Process*, vol. 25, no. 7, pp. 711–741, 2013. [Online]. Available: http://dx.doi.org/10.1002/smr.1560

[37] R. Kazman, J. Asundi, and M. Klein, "Quantifying the costs and benefits of architectural decisions," in *Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on*, May 2001, pp. 297–306.

[38] H. Markowitz, "Portfolio selection," *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952. [Online]. Available: http://www.jstor.org/stable/2975974

[39] D. Krajnc and P. Glavi, "How to compare companies on relevant dimensions of sustainability," *Ecological Economics*, vol. 55, no. 4, pp. 551 – 563, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921800904004513