Refactoring Society: Systems Complexity in an Age of Limits

Barath Raghavan
ICSI
barath@icsi.berkeley.edu

Daniel Pargman
KTH
pargman@kth.se

ABSTRACT

Research in sociology, anthropology, and organizational theory indicates that most societies readily create increasingly complex societal systems. Over long periods of time, accumulated societal complexity bears costs in excess of benefits, and leads to a societal decline. In this paper we attempt to answer a fundamental question: what is the appropriate response to excessive sociotechnical complexity? We argue that the process of refactoring, which is commonplace in computing, is ideally suited to our circumstances today in a global industrial society replete with complex sociotechnical systems. We further consider future directions for computing research and sustainability research with the aim to understand and help decrease sociotechnical complexity.

CCS Concepts

•**Applied computing** → **Law, social and behavioral sciences;** •**Software and its engineering** → *Software post-development issues;*

Keywords

Sustainability; refactoring; complexity

1. INTRODUCTION

Computing systems can be seen as the latest instance in a long chain of sociotechnical developments that have increased social complexity at an accelerated pace [4, 23, 24]. Today global industrial society is particularly dependent upon and highly mediated by many computing systems. Over the past several decades, the growth of such systems have by and large been a boon as they have enabled a revolution in the way we communicate, work, and live. During this time, computer scientists in particular have contributed much to the development and spread of systems that have subtly, but unmistakably, transformed global society. Computing systems have also amplified sociotechnical complexity and accelerated previous trends far more than prior technologies because of the inherent complexity of networked systems and the interlinking of previously independent systems [14].

LIMITS '16, June 08-10, 2016, Irvine, CA, USA

ISBN 978-1-4503-4260-5/16/06…$15.00

DOI: http://dx.doi.org/10.1145/2926676.2926677

While societal complexity generally, and complex computing systems specifically, brings benefits, it also imposes costs. While there are some exceptions [3, 18, 28, 34, 35, 40, 41], the sustainable computing community seldom examines large-scale and long-term costs associated with the systems we develop. Physical infrastructure, such as the power grid or Internet backbone, provides an easy to understand example of how complex systems have both benefits and costs. The costs are often not well understood and pertain to not only the expense of construction and maintenance, but also the long-enduring commitments the infrastructure entails (including concomitant socioeconomic costs of required social arrangements, of energy sources, and of adjunct technologies each with their own costs). The benefits of such systems are better understood, including the offsetting of costs due to the prior systems the new infrastructure obviates.

As Joseph Tainter argues in his classic work [37], societies generally solve problems in ways that increase complexity (e.g., building additional and/or more complex sociotechnical systems). Indeed, it can be argued that civilization itself, including traditions of great art and writing, "are epiphenomena or covariables of social, political, and economic complexity" [37]. For a time, increasing complexity produces benefits in excess of costs, but complexity is also subject to declining marginal returns, as Tainter depicts with a simple but profound diagram shown in Figure 1, which we refer to as the Tainter curve.

The Tainter curve indicates that simple solutions with major benefits will eventually be followed by more complex solutions with modest incremental benefits. At some point, increasing complexity fails to yield net benefits. At that point—point C2 on the Tainter curve—a wise approach would be to control or preferably reduce complexity. Yet seldom are such circumstances recognized or measures taken, and Tainter (as well as Diamond [10]) showed as a result how many historical civilizations continued to increase complexity until the attendant costs drove these societies into collapse, a phase of rapid decline in societal complexity.[1] To an anthropologist such as Tainter, "rapid decline" can refer to a period of decades or centuries. Decline in social complexity includes a variety of phenomena such as less centralised political and economic control, reduced use of natural resources, decreased economic and occupational specialization of individuals, groups and territories, and, a reduction of population size and density.

Even if a society were wise enough to stop further increases of complexity at the point of zero marginal benefit, the Tainter curve is not static and the costs of sociotechnical complexity can become

[1]That the arrival of a society at point C2 is not widely recognized appears to strongly relate to Daly's observations about "uneconomic growth", which is not apparent in today's dominant neoclassical economic thinking [9].

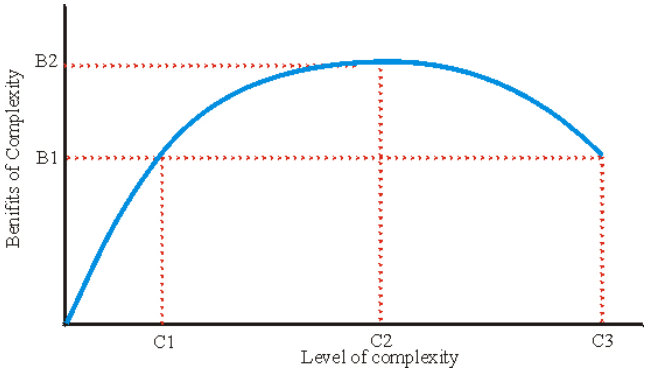

Figure 1: The Tainter curve, depicting the benefits of complexity to a society relative to the degree of complexity [37].

harder to bear over time due to decreased access to essential resources. The equivalent example from the biological world is the decrease in carrying capacity of an ecosystem due to a population being in overshoot of that carrying capacity, in effect degrading the functioning of the ecosystem itself for the long term. More specifically, human societies today are dependent upon the broader ecosystem from which they extract resources (e.g., fossil fuels and minerals) and dump wastes (e.g., greenhouse gases). Ecological limits, which shrink the availability of resources and impose costs due to environmental calamities, will over time reduce the ability of societies to maintain a given level of complexity. In an age of the ecological limits that we face today [42], the declining availability of non-renewable resources (e.g., fossil fuels and minerals) and the costs from climate change and other ecosystem damage threaten to make today's complexity more difficult to maintain, to say nothing of further increasing societal complexity.

We argue that it has come time to *refactor society*. Refactoring in computing is the process by which the internal structure of a piece of software or larger system is improved, typically reducing its complexity, while still retaining its core (externally-visible) functionality [21, 26]. Thus refactoring society would entail simplifying large-scale sociotechnical systems while retaining all or most of their benefits. We propose the introduction of refactoring as a goal in development practices for computing generally and as a core tenet of sustainable computing in particular.

In the rest of the paper we define more precisely, following Tainter, the notion of complexity that concerns us, and consider where in Tainter's curve we are today. We then examine complexity in the context of computing systems specifically, and consider their impact on the complexity of global society today. Then we examine how we might go about refactoring society and describe several general principles and approaches that can help in this effort, building upon refactoring approaches that have proved useful for computing systems. Finally, we propose a set of open research questions for further study.

2. ON SOCIETAL COMPLEXITY

Studies of social complexity compare simpler societies (for example hunter-gatherer bands, tribes, or chiefdoms) with more complex societies (for example kingdoms or states), where the former are small, homogeneous and minimally internally differentiated and the latter are large, heterogeneous and highly internally differentiated. Human societies, like individuals, are dependent upon a constant supply of energy. Compared to small groups of self-sufficient foragers, complex societies are more costly to maintain and require more energy per capita. All great civilizations in the past have been forced to manage on a flat energy budget, based on solar energy and the crops growing in the fields (sometimes receiving temporary subsidies through conquest of neighbors). Tainter argues that rather than being primary, the demand for greater energy flows and their subsequent use are consequences of increased complexity [38].

Tainter and Patzek suggest that the alternative to making do on a fixed, solar-based energy budget is the Western strategy of the past 200 years: to dig for new, inexpensive, abundant sources of energy that can subsidise ever-increasing levels of sociotechnical complexity [39]. Fossil fuels (coal, oil, gas) currently constitute 5/6 of all energy consumed globally [5]. However, these fuels are nonrenewable and should we fail to further increase current levels of energy production, we too have to manage on a flat (or worse, shrinking) energy budget [13]. The only alternative as suggested by Tainter and Patzek is to decrease the costs of complexity by preemptively, deliberately, and systematically simplifying society. The costs of doing so would be that we have to give up some of our accustomed ways of life.

While the aim of this paper is to argue for the reduction of complexity, it is not the case that complexity is intrinsically bad. Complexity brings benefits and complexity has costs. The benefits and the costs should be considered in relation to each other, but they often are not. Increased complexity is the typical outcome of solving problems and it is either useful and affordable or it is not. Societies (indirectly) choose to increase complexity because adding complexity often solves existing problems and yields net benefits; solving problems through the use of sociotechnical systems, even problems that urgently "need to be solved", will typically (often inadvertently) increase complexity.

Economic growth, as commonly understood today, typically involves the increase of complexity; growth is measured by an increase in Gross Domestic Product. Each transaction using money, especially in highly-structured economies, takes place via some sociotechnical system, and growth typically means additional sociotechnical systems through which (or for which) money is transacted. At some point, even while growth continues in absolute terms, it can become "uneconomic", as noted by Daly, and result in a declining quality of life [9].

Measuring societal complexity is not an easy undertaking. Nor is it a precise or uncontentious process, though it is an easier task today than in past decades, as more socioeconomic arrangements are part of the formal economy and thus can be easily included in the analysis. Nevertheless, we aim to sidestep the need to quantify societal complexity, as a precise characterization is unnecessary for the purposes of this paper. Instead, our focus is on what region of the Tainter curve we find ourselves today. While even this cannot be determined with any certainty, we argue that it would be prudent to consider the consequences of "overshoot"—of taking on more complexity than is beneficial.

Specifically, we consider four points on the Tainter curve, three of which are labeled in Tainter's original diagram (Figure 1):

- C1: This position is one that simpler societies occupy and is marked by a huge potential increase in benefit for each further problem solved and for each increase in complexity. That is, each unit of economic growth (and attendant complexity) produces large societal benefits.

- C1.5: This unmarked position somewhere between C1 and C2 in Tainter's diagram is the one global society has remained in for at least a few decades in which each increase

in societal complexity produces benefits but also non-trivial costs. The benefit from each subsequent unit of growth or complexity is declining.

- C2: This position is the one of maximum benefits. Here complexity is either too great to find net beneficial changes and/or the impact of ecological limits is such that each increase in complexity results in negative net societal benefits.
- C3: This position is one of overshoot and impending collapse. Here society has mired itself so deeply in excess complexity that rapid societal simplification—what Tainter defines as societal collapse—is the likely result.

We believe that C1 and C3 are unlikely to accurately reflect the state of today's global industrial society. Instead, we will assume in this paper that industrialized nations are approximately at C2 on the Tainter curve. Thus it is prudent for us to consider approaches to prevent overshoot and instead reign in complexity. Indeed, many scientists examining the sustainability of industrial society today have implicitly concluded that we have already passed C2, making the need to decrease complexity more urgent still [2, 7, 42].

3. SOCIOTECHNICAL SYSTEMS AND COMPLEXITY

Complexity in technical systems can be quantified through a variety of means. At the purest level, researchers in complexity and information theory have developed means to measure complexity (e.g., Kolmogorov complexity [15], computational complexity [27]), though these mathematical formulations, while precise and analyzable, are too narrow for us.

One step removed from such theoretical formulations is Ratnasamy complexity, a proposed metric for the complexity of a network protocol or system, such as an Internet routing protocol [8, 36]. Ratnasamy's metric attempts to capture the amount of information that must be stored and then conveyed across a distributed series of nodes (e.g., Internet routers); in this, both the origin of information and the distance the information must travel, conveys the complexity of a protocol.

We do not believe it is possible to quantify the complexity of large-scale sociotechnical systems, since any theoretical framing that enables quantification would inevitably ignore or abstract away important real-world (e.g., human- or eco-centric) details that cannot be captured in the model. Nevertheless, social scientists have developed approaches for understanding complex human systems [22, 29] and systems theorists have developed approaches for modeling and understanding complex human and ecological systems in ways that capture relevant factors and enable reasoning [19, 20].

Industrial societies are replete with complex sociotechnical systems that contribute to most of their complexity today. Many of these systems are marked by significant use of abstraction and indirection, two of the core principles that enable much of computing design [32]. In examining abstraction and indirection, and thinking about sociotechnical systems like we do computing systems, then, we might be able to uncover a coarse understanding of the complexity of sociotechnical systems.

The Tainter curve is true not just for societies but also for sociotechnical systems. In computing we are already very familiar with the Tainter curve: we understand that the complexity-benefit curve arises in many computing systems, especially in complex software systems where additional complexity often yields few benefits. However, when augmenting such systems it is often easier to add additional complexity to solve an existing problem rather than do otherwise, ultimately yielding a worse system after crossing the top point of the curve. We are also too familiar with systems that have gained an undesirable degree of complexity: *spaghetti code*, *feature creep*, and *software bloat*. The failure of complex software (and hardware) systems due to unmanageable complexity is widely documented and the last few decades are littered with examples. That adding complexity—in the form of additional people and resources—can cause the demise of a project that is already struggling is one of the key findings of Brooks's classic work on large-scale computing projects [6].

Given the dependence of wealthy industrialized nations on computing systems today, the complexity (and failings) of large-scale computing systems no longer are isolated from the functioning of society as a whole. Writing more than 30 years ago, Perrow argued that accidents should be regarded as an integrated and inevitable effect of complex technological systems [30]. In the intervening decades, software and complex computing systems have become a central source of both societal solutions as well as woes, from ensuring the availability of basic infrastructure (e.g., the power grid [31], the Internet [17]), providing access to government services (e.g., the U.S. healthcare.gov website [16]) and the functioning of military machinery (e.g., the F-35 warplane [1]).

The solution to too much complexity in the context of individual large-scale computing and/or software systems is to attempt to reduce or manage that complexity. There is a general recognition among software engineers that it is not possible to precisely identify the point at which complexity has exceeded a critical threshold, but rather that when the system is in the danger zone, attempts should be made to remedy the situation before it gets worse.

The solution in computing when faced with such undesirable complexity is to refactor. Refactoring entails identifying and leveraging opportunities for code reuse, for removing obsolete or unnecessary code that adds no value but increases complexity, for simplifying complex code abstractions, for reducing the number of layers of nesting or indirection employed, and for many more similar techniques. The resulting system, if refactored effectively, will not only be more robust and easier to improve and maintain, but will result in lower costs (both financial and mental toll) for software engineering teams. Some refactoring requires the elision of previously desired functionality in a deliberate simplification that may result in fewer features but a more streamlined system.

4. REFACTORING SOCIETY

When the system in question is a sociotechnical system rather than a purely technical system, the task before us is to *refactor society*. The case we make here is that there is significant value and reason to attempt to decrease the complexity of society in general and sociotechnical systems in particular. Sustainability research generally aims to find new approaches that enable society to preserve ecological function for use by future human and non-human use, and to thus endure as a society. If, as Tainter and others argue, complex societies tend to burden themselves with surplus complexity and as a result collapse (which by definition is not sustainable), attending to the problem of surplus complexity is in fact a core challenge for sustainability research.

A core assumption of this approach is that sociotechnical structures in a complex society can be mapped like a complex technical system, and complexity can be stripped out while retaining all or most of the benefit. We do not know if this can be achieved in general, and even in specific cases it is a challenging undertaking.

There are two ways computing researchers and engineers can help towards this important goal. First, *do no harm*; that is, we should aim, at the minimum, to not increase the complexity of so-

ciety through the systems we build, and ideally decrease complexity (due to the replacement of existing more complex alternatives). Second, *refactor*; that is, we should explicitly aim to redesign existing systems to reduce societal complexity, and this should be considered a worthwhile goal of computing research and engineering.

4.1 When Designing a System

When designing a system, the challenge of understanding whether the system does no harm, let alone decreases societal complexity is a daunting one. Thus each evaluation will necessarily be limited by the time and resources available to the system's designer(s). Nevertheless, we believe that it is possible to ascertain the basic complexity of a system and its impacts.

The complexity analysis a designer should perform can be thought of as both being inside out and outside in. That is, the analysis should first proceed from the system in its most basic manifestation (e.g., a piece of software or hardware) and then consider the dependencies and interacting sociotechnical elements moving outward. Then the analysis should consider large-scale social system structures in which the system might play a role, and identify how those existing structures are affected by the new system. The approach undertaken here is likely similar in nature to life-cycle analysis, though with a broader aim.

All new systems will necessarily possess some embodied complexity. All computing systems have a physical basis, and this physical basis entails dependence upon a complex global supply chain for the manufacture, distribution, and disposal of computing components. In addition, most modern computing systems depend and place demands upon existing infrastructure such as the electric grid, the Internet, datacenters, and far more [33,34]. These direct dependencies on energy, resources, infrastructure, human engineers and technicians, and so forth are first-order impacts. It is rare for a new system to decrease the complexity of society when these first-order impacts are considered, as no computing system is without a footprint.

Second-order impact analysis considers the changes in adjunct systems caused by the system in question. For example, a system that trains users how to build a smartphone from spare parts enables them to no longer have to buy new smartphones and therefore the burden on the complex systems entailed in manufacturing smartphones are reduced. Considering third-order and beyond is difficult, but sometimes worthwhile when there are possible distant systems that are likely to be impacted by the ripple effect of the new system in question.

An important approach for new systems is to aim for self-obviation [41]. A self-obviating system is one whose benefits can remain even after the system is gone (and with it, its complexity). Such systems have the potential to increase complexity for a time-limited duration, helping ensure that society's overall complexity is not permanently increased.

4.2 When Refactoring

When refactoring, it is not often the case that the designer or engineer is building an entirely new system; in fact, we believe that the principles of undesign are likely to be highly relevant here [3]. Such work will often begin from a deep analysis of the sociotechnical structures that require simplification.

One of the core aims of refactoring is to beat the complexity-benefit tradeoff. This is a challenge in itself, as Tainter's analysis indicates that most or all work to solve problems inherently increases complexity. However, clearly complexity does not always increase: collapse is a process in which complexity chaotically and rapidly decreases. Our aim is instead to achieve a deliberate and controlled decrease and/or moderation of complexity.

4.3 Aiming for the Ideal

The goal of decreasing societal complexity is in some sense to find the "sweet spot" where complexity is low enough that additional complexity still yields net benefits, and help ensure that society remains as close to that point on the Tainter curve as possible. The sweet spot will not be the point of maximum benefit, but rather to the left, at a point of lower complexity but nearly as high benefit. That is, the ideal point is the point of diminishing returns, when each increment of complexity yields little additional benefit.

What we desire is a dynamic equilibrium, in which society and sociotechnical systems continue to change but we ensure that total complexity does not grow. While in the context of this abstract discussion it may not be immediately apparent at what point the diminishing returns are worthwhile, it is the case that in specific industries this tradeoff is somewhat understood (e.g., in the fossil fuels industry, the energy return on energy invested must generally be above some ratio, such as 3:1, before an unsubsidized fossil energy source is worth extracting).

The identification of a potential sweet spot is a design tool, and is less about the spot itself. For a specific context, it requires the designer to evaluate both increasing complexity (and looking at the potential benefit) and decreasing complexity (and look at the potential loss of benefit), and in doing that we are forced to critically evaluate the complexity-benefit tradeoffs.

5. APPLYING EXISTING TECHNIQUES

While some of the techniques used for refactoring of software systems are necessarily specific to such systems, focused on arcane details unique to computer programs, many of them can be generalized to non-software systems. While their generality is not easy to confirm without years of testing, here we consider how to transform standard techniques of refactoring known to computer scientists to the refactoring of larger complex systems.

In his classic book on the subject, Fowler lists indications that a piece of software needs to be refactored [11]. While there are certainly other techniques in software refactoring we could look to as a guide for societal refactoring, Fowler's list suffices for the brief discussion we include below. We note in advance that none of these observations, either from Fowler or our generalizations of them, are unique or novel, nor would they be surprising to most engineers. Despite this, we think that enumerating the possibilities is valuable, as they provide a checklist approach that can help turn the complex task of refactoring into a procedural one.

Specifically, beyond subjective indicators, Fowler suggests the following as signs that software should be refactored, which we now consider in its software context and attempt to extrapolate to a broader context:

1. **Duplicated code** typically manifests as slightly different variations of the same code in two different places in the system. *Generalized,* sociotechnical systems that duplicate some aspects of their core behavior should be consolidated.

2. **Long methods** often increase the complexity of a program because the difficulty in understanding a long method (one that attempts to do too many things) is superlinear in its length. *Generalized,* sociotechnical systems that have subsystems that consolidate many different (sometimes disparate) tasks should be divided into easier to understand pieces.

3. **Large classes** hold too much data and attempt to implement too many interfaces, and sometimes also have internal redundancy. *Generalized,* sociotechnical systems that have too many small subsystems, some of which overlap, should have the subsystems merged.

4. **Long parameter lists** are cases in which too many things are being specified by the caller of a method, rather than being internal to the method and/or class. *Generalized,* sociotechnical systems should limit the amount of dependence they have on (and the degree to which they accept) external inputs when attempting to achieve their tasks.

5. **Divergent changes** are cases in which to make one modification to a class requires certain changes, but a different modification requires very different changes, indicating that the class is overspecialized and/or internally inconsistent. *Generalized,* sociotechnical systems should divest specialization for specific use cases to other associated systems rather than internally specializing.

6. **Shotgun surgery** involves making many small changes across many classes to achieve some specific type of change, which is hard to do correctly because some of these small changes may be missed. *Generalized,* common types of effects of goals that previously required small changes across many systems should be consolidated into one simple sociotechnical system that embodies all the required changes.

7. **Feature envy** involves classes that rely more heavily on the methods and/or data of other classes than their own, indicating that the methods and/or data are not appropriately organized. *Generalized,* sociotechnical systems that overly depend upon interaction with another should be recompartmentalized to include the relevant pieces of functionality that change together.

8. **Data clumps** are groups of data that tend to co-exist in classes, and should be made into their own class. *No natural generalization.*

9. **Primitive obsession** involves overuse of primitive data types instead of small classes. *No natural generalization.*

10. **Switch statements** involves overuse of switch statements to make decisions instead of using polymorphism. *No natural generalization.*

11. **Parallel inheritance hierarchies** occur when subclassing one class requires subclassing of a parallel class. *Generalized,* sociotechnical systems that have parallel counterparts can be made to directly interface in practice rather than codify the communication they make with each other (and their corresponding subunits).

12. **Lazy classes** are classes that don't do very much of use, especially if some of their functionality has been moved by refactoring; such classes should be eliminated. *Generalized,* vestigial sociotechnical systems should be periodically identified and dismantled.

13. **Speculative generality** occurs when extra general functionality is added in advance of it being needed, especially with extra indirection and abstraction. *Generalized,* extra layers of abstraction or indirection should be periodically identified and their current (rather than speculative) use should be the guide for whether they are needed.

14. **Temporary fields** are fields that are not always used in a class. *No natural generalization.*

15. **Message chains** occur when a method asks objects, and in turn asks those objects for other objects; these can be eliminated by encapsulating the delegation. *Generalized,* sociotechnical systems should be built to have to know as little as possible about the inner workings and interaction of the systems upon which they depend, providing greater independence and flexibility.

16. **Middle man** is a problem that occurs when too much delegation is going on, and much of that delegation is pass through. *Generalized,* excessive delegation and intermediaries should be removed from sociotechnical systems where possible.

17. **Inappropriate intimacy** occurs when classes know or need to know too much about the inner workings of another class. *Generalized,* sociotechnical systems should be designed to know only about the public interfaces they have with one another, and even simpler, have only a single-direction relationship between systems.

18. **Alternative classes with different interfaces** are cases in which similar methods appear in different classes with different signatures. *Generalized,* sociotechnical systems that have some common functionality should have that functionality standardized so as to make their invocation simpler for those on the outside.

19. **Incomplete library classes** are an issue when the standard library does not provide sufficient basic functionality. *No natural generalization.*

20. **Data classes** are classes that hold data and do nothing with that data. *No natural generalization.*

21. **Refused bequests** are classes that do not use most of what they inherit, indicating that they inherited from the wrong class. *Generalized,* subinstances of a sociotechnical system that do not need to support the behavior of the parent system should instead have the parent system move the non-general functionality to a sibling system.

22. **Comments** are sometimes overused in badly written software to cover for poor design. *Generalized,* an indication that a sociotechnical system is too complex is that the users, operators, or engineers need far too much explanation to understand how to use, run, or change the system.

6. RESEARCH QUESTIONS

We believe that the idea of refactoring society can fundamentally realign not only the designs we devise but even the research we deem worthy of pursuit. Next we consider questions, issues, design ideas, and possible leads for further work on refactoring society. Some of the directions we suggest are narrow in scope, possibly of the size of a Masters thesis, whereas others are of large scope and could be suitable for a PhD dissertation.

1. Extend Ratnasamy Complexity [8, 36] to analyze and quantify the complexity of a broader class of sociotechnical systems.

2. Identify and characterize the sweet spot in systems for the Internet of Things (IoT), life logging, quantified self, or similar complex technical or sociotechnical systems.

3. Extend the metaphor of refactoring. Examine bureaucratic systems in society as code (e.g., laws, government or corporate structures, etc.), since they are codified (i.e., they are specified and do not change frequently). Apply ideas from code refactoring to these systems.

4. Extend the metaphor of refactoring further still, and consider the fact that many bureaucratic systems are now actually being embodied in actual code (e.g., administrative computing systems that represent procedural actions), giving a more precise means for refactoring. Apply ideas from code refactoring to these systems.

5. Develop a methodology for diagramming or mapping any sociotechnical system to help in its analysis and possible refactoring.

6. Examine Meadows's arguments about leverage points in systems [20] to identify which leverage points are most appropriate for decreasing sociotechnical complexity, especially within specific contexts or industries.

7. Extend the connection with Meadows's leverage points discussion [20] by directly relating these arguments to specific techniques in the software refactoring literature.

8. Model complex sociotechnical systems using Meadows's systems theory.

9. Study connections between Odum's energy systems diagram approach and the modeling of sociotechnical system complexity using other approaches (such as those above) [25].

10. Evaluate whether one metric (e.g., Odum's concept of solar emJoules [25]) or a small number of metrics (e.g., solar emJoules plus Ratnasamy complexity) yield useful results in categorizing and describing the complexity of large-scale systems.

11. Explore connections between specific disciplines in which efforts to simplify and/or cope with complexity have yielded significant beneficial results (e.g., checklists in medicine [12]).

12. Identify means for testing refactored sociotechnical systems (akin to the test suites that are leveraged during software refactoring to ensure that the core functionality continues to work).

13. Study how and whether self-obviating systems convey their complexity onto other systems so that their (complexity/benefit) impact remains even after the system is removed [41].

14. Study the relationship and causality between complexity and energy use in sociotechnical systems [38].

7. CONCLUSION

We face a choice today: we can continue to increase societal complexity and likely suffer the same fate as past societies that did not respond until far too late, or we can step back and choose a more sustainable path. While this choice has been known for some time, we hope that our articulation of the challenge of refactoring sociotechnical systems, and thus society, provides a concrete approach that is distinct from the traditional sustainability approaches.

8. REFERENCES

[1] D. Axe. World's Most Expensive Jet Somehow Gets Worse. *The Daily Beast*, January 28, 2016.

[2] A. D. Barnosky, E. A. Hadly, J. Bascompte, E. L. Berlow, J. H. Brown, M. Fortelius, W. M. Getz, J. Harte, A. Hastings, P. A. Marquet, et al. Approaching a state shift in earth's biosphere. *Nature*, 486(7401):52–58, 2012.

[3] E. P. Baumer and M. Silberman. When the implication is not to design (technology). In *Proceedings of ACM CHI*, 2011.

[4] J. R. Beniger. *The control revolution: Technological and economic origins of the information society*. Harvard University Press, 1986.

[5] BP. Statistical Review of World Energy, June 2015.

[6] F. P. Brooks. *The mythical man-month*. Addison-Wesley, 1975.

[7] W. R. Catton. *Overshoot: The ecological basis of revolutionary change*. University of Illinois Press, 1982.

[8] B.-G. Chun, S. Ratnasamy, and E. Kohler. Netcomplex: A complexity metric for networked system designs. In *Proceedings of USENIX/ACM NSDI*, 2008.

[9] H. E. Daly. Uneconomic growth: in theory, in fact, in history, and in relation to globalization. *Clemens Lecture Series. Paper 10*, 1999.

[10] J. Diamond. *Collapse: How societies choose to fail or succeed*. Penguin, 2005.

[11] M. Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley, 1999.

[12] A. Gawande. *The checklist manifesto: how to get things right*. Metropolitan Books, 2009.

[13] C. A. Hall and J. W. Day. Revisiting the limits to growth after peak oil. *Am Sci*, 97(3):230–237, 2009.

[14] T. Homer-Dixon. *The upside of down: catastrophe, creativity, and the renewal of civilization*. Island Press, 2010.

[15] A. N. Kolmogorov. On tables of random numbers. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 369–376, 1963.

[16] S. Levy. America's Tech Guru Steps Down—But He's Not Done Rebooting the Government. *Wired*, August 28, 2014.

[17] R. Mahajan, D. Wetherall, and T. Anderson. Understanding bgp misconfiguration. In *Proceedings of ACM SIGCOMM*, 2002.

[18] J. C. Mankoff, E. Blevis, A. Borning, B. Friedman, S. R. Fussell, J. Hasbrouck, A. Woodruff, and P. Sengers. Environmental sustainability and interaction. In *CHI'07 extended abstracts on Human factors in computing systems*. ACM, 2007.

[19] D. Meadows, J. Randers, and D. Meadows. *The limits to growth: the 30-year update*. Chelsea Green, 2004.

[20] D. H. Meadows and D. Wright. *Thinking in systems: A primer*. Chelsea Green Publishing, 2008.

[21] T. Mens and T. Tourwé. A survey of software refactoring. *Software Engineering, IEEE Transactions on*, 30(2):126–139, 2004.

[22] H. Mintzberg. The nature of managerial work. 1973.

[23] L. Mumford. *Technics and civilization*. Harcourt, Brace and Company, 1934.

[24] L. Mumford. *Technics and Human Development: The Myth of the Machine*. Harvest Books, 1971.

[25] H. Odum. *Environmental accounting: emergy and environmental decision making*. John Wiley & Sons, 1996.

[26] W. F. Opdyke. *Refactoring: A program restructuring aid in*

designing object-oriented application frameworks. PhD thesis, PhD thesis, University of Illinois at Urbana-Champaign, 1992.

[27] C. H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[28] D. Pargman and B. Raghavan. Rethinking Sustainability in Computing: From Buzzword to Non-negotiable Limits. In *Proceedings of ACM NordiCHI*, 2014.

[29] C. N. Parkinson and R. C. Osborn. *Parkinson's law, and other studies in administration*, volume 24. Houghton Mifflin Boston, 1957.

[30] C. Perrow. Normal accidents: Living with high risk systems, 1984.

[31] P. Pourbeik, P. S. Kundur, and C. W. Taylor. The anatomy of a power grid blackout. *IEEE Power and Energy Magazine*, 4(5):22–29, 2006.

[32] B. Raghavan. Abstraction, Indirection, and Sevareid's Law: Towards Benign Computing. In *Proceedings of LIMITS*, 2015.

[33] B. Raghavan and S. Hasan. Macroscopically Sustainable Networking: On Internet Quines. In *Proceedings of LIMITS*, 2016.

[34] B. Raghavan and J. Ma. The energy and emergy of the internet. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, page 9. ACM, 2011.

[35] B. Raghavan and J. Ma. Networking in the Long Emergency. In *Proceedings of the ACM SIGCOMM Workshop on Green Networking*, 2011.

[36] S. Ratnasamy. Capturing complexity in networked systems design: The case for improved metrics. In *Proceedings of HotNets*, 2006.

[37] J. Tainter. *The collapse of complex societies*. Cambridge University Press, 1990.

[38] J. A. Tainter. Resources and cultural complexity: Implications for sustainability. *Critical reviews in plant sciences*, 30(1-2):24–34, 2011.

[39] J. A. Tainter and T. W. Patzek. *Drilling down: The Gulf oil debacle and our energy dilemma*. Springer Science & Business Media, 2011.

[40] B. Tomlinson, E. Blevis, B. Nardi, D. J. Patterson, M. Silberman, and Y. Pan. Collapse Informatics and Practice: Theory, Method, and Design. *ACM Transactions on Computer-Human Interaction*, 2013.

[41] B. Tomlinson, J. Norton, E. Baumer, M. Pufal, and B. Raghavan. Self-obviating systems and their application to sustainability. In *Proceedings of the iConference*, 2015.

[42] M. Wackernagel, N. B. Schulz, D. Deumling, A. C. Linares, M. Jenkins, V. Kapos, C. Monfreda, J. Loh, N. Myers, R. Norgaard, et al. Tracking the ecological overshoot of the human economy. *Proceedings of the National Academy of Sciences*, 99(14):9266–9271, 2002.