Collapse (& Other Futures) Software Engineering

Birgit Penzenstadler, Ankita Raturi,
Debra J. Richardson,
M. Six Silberman, Bill Tomlinson
University of California, Irvine

ABSTRACT

Current software engineering efforts typically rely heavily on industrial infrastructure. In a situation of civilizational collapse - which various researchers have suggested could ensue in the next century via an assortment of environmental, economic, and/or social pathways - such infrastructure may become less reliable. Previous research has offered some thoughts about what new forms of software may be relevant in the context of collapse. However, those papers did not consider how these new kinds of software would arise. Building on previous work in software engineering for sustainability (SE4S), information and communications technology for development (ICT4D), and collapse informatics, this paper explores how various forms of civilizational collapse would affect the software development process.

Categories and Subject Descriptors

D.2.0 [**Software Engineering**]: software development process;

General Terms

Management, Documentation, Design, Economics, Reliability.

Keywords

Software development process, future scenarios, sustainability, collapse, standards.

I. INTRODUCTION

Software Engineering (SE) has long benefitted from Moore's law – with progress, faster computers, better networks, faster communication, and instant access, SE moved from programming on punch cards to live collaborative software development with near instant feedback. It generally is assumed that this kind of increasingly rich context of software systems and surrounding infrastructure will continue. Indeed, the evolution to increasingly smart phones, tablets, and corresponding apps seems to confirm this. What if, however, such progress did *not* occur? What if under certain scenarios of how society evolves, there actually is a decrease in processing speed, storage capacity, and other aspects of computing? We would need to rethink not just the systems we build, but also how we build those systems. While this may sound far-fetched, such alternative scenarios to constant progress are being considered by other research disciplines. Researchers in anthropology, geography, and other fields have suggested that the collapse of global industrial society is not implausible. Scientific understandings of collapse often suggest that institutional infrastructures will not cease all-at-once; rather, they will gradually become less effective and their services will grow ever more intermittent [4]. As this transformation occurs, new forms of software will likely become more relevant, and certain existing processes for creating software will become less viable.

Software Engineering researchers have not really worried about the potential of collapse. The green computing vision - broadly construed - begins to point in that direction, if ever so slightly, as does Software Engineering for Sustainability (SE4S) [1,2]. This paper seeks to explore the effects that various forms of civilizational collapse could have on the software development process.

II. BACKGROUND

There are currently more than 1M software developers in the US alone [5], and hundreds of thousands more in other parts of the world. These developers design and build the software systems that operate much of industrial civilization. Software engineering is the field that studies and improves how these developers do the work they do, often working in large teams to develop systems that will be used by millions or even billions of clients.
Software engineers currently use a range of processes, from linear, iterative and evolutionary to agile as well as hybrid approaches in between. These software engineering processes rely on the ongoing availability of a range of different kinds of infrastructure, and are critical in the operation of those infrastructures. Of the Department of Homeland Security's list of Critical Infrastructure Sectors [6], communication, energy, and IT are perhaps the most heavily involved with software engineering; however, many other sectors are involved as well. Because of the interdependencies among these infrastructures [7], intermittency or failure of many different sectors could compromise current software engineering processes. This paper seeks to explore how software engineering could be transformed by these interruptions.

III. IMPACT ON CURRENT PRACTICES

There are a number of impacts that a collapse scenario would have on current practice, inter alia:

A. Software requirements

The new roles software and related technologies will have in a collapsing or post-collapse civilization would determine new requirements for existing and new systems yet to be developed. Depending on the nature of collapse, stakeholders and audiences, along with their demands, would change. In the case of economic collapse, there may be different requirements for financial systems, bartering systems, and user auction services (like eBay and Craigslist). In case of societal collapse, yet-another-dating-app might not be needed, but new requirements would arise for functional communication software. We would need to rethink the function of software. For the example of the bartering system, a requirement would be that it has to run on a resilient server with redundant data storage because energy supply might be interrupted in such a future of scarcity.

B. Software design

There is a design-for-environment movement in software engineering and systems engineering where systems are designed to be more environmentally friendly, but if we were designing for collapse, we would have different constraints. These constraints would, for example, arise from societal changes, like a decentralized government. Integrating the human-centered engineering movement and environmental engineering, we can develop a coherent set of methods, metrics, and tools to design for collapse. In particular, collapse could tie together societal, economic, and environmental concerns.

On the technical networking level, one instance is currently popular architectures (including model-view-controller (MVC), client-server, pub-sub) are not necessarily collapse-resilient: they focus on performance and throughput, cost, effort, and sometimes efficiency. Under different collapse scenarios, different software architectures would become prominent. In case of infrastructural collapse, architectures (e.g., peer-to-peer) amenable to mesh networks and alternative networking protocols would be most useful. Mesh networks are set up a like large peer-to-peer system but are networking completely distributedly. They are often used in alternative Internet scenarios.

C. Software construction

Software engineering is currently a highly collaborative, global endeavor. As this changes in different collapse scenarios, development tools will need to evolve accordingly. Under many collapse scenarios, collaborative software development tools like rackspace and GitHub may become intermittent and unreliable. There are many open source tools that developers of the future can use to setup alternative, likely local, software development environments (e.g., set up a local LAMP server to support a small development team as opposed to using Linode). Once again situating the ideas from green software engineering environments, for example energy efficient data centers, and resilient distributed knowledge management, we can begin to address additional issues like intermittent energy outages.

D. Software quality

Depending on the collapse scenario, there would be a reprioritization of software quality attributes. For example, when thinking about reliability, resilience to power outages may outweigh business performance. In simple words, no online shop will make money, or have to worry about its business performance, if its offline – assuming we still have online shops. Efficiency concerns would be with respect to resources rather than finances. The types of software testing conducted would similarly be affected. Usability testing, for example, may take a back seat to robustness testing.

E. Software maintenance

Current demands on software tend to require efforts to scale up systems, respond to increased traffic, and (recently) reduce energy consumption. In collapse scenarios, software maintenance efforts would be adaptive, involving the retrofit of high priority existing software systems to survive the changes. However, the systems would have to be modified to cope with changes in the physical environment, not just the software environment. The disappearance of frameworks will require us to reverse engineer our systems and that will enable us to develop systems that are on more basic platforms. We would also have to prioritize software systems based on community needs: expend effort to maintain medical control systems versus online gaming systems. Furthermore, we might not be able to buy replacement parts for our computers anymore. If we have to maintain our computers at all effort we have to scale down and retrofit ever-increasing computing demands to platforms that don't grow along to simpler, leaner platforms. We might not be able to print circuit boards anymore in such a scarce future, but we could still make PCBs (programmable circuit boards) as we used to make them 30 years ago.

F. Software engineering economics

In case of economic collapse, the current 'business' of software could change in at least three ways. One way would be that we all agree that free and open source software may become the norm, with people developing systems openly, voluntarily, and freely sharing knowledge in the commons. Secondly, potentially collapse-resilient currencies such as Bitcoin may become the core of a new software economy. Some organizational structures (e.g., cooperatives, B Corporations) may be more well-positioned than others (e.g., public corporations whose management is expected to maximize short-term shareholder value) to organize and support the development and maintenance of software that responds to user needs in collapse scenarios, and to anticipate such needs pre-collapse [3]. Thirdly, a script-economy may arise, where the kids who write small bash scripts that work on tiny Linux boards, take over as the script-mafia. Bash scripting, as a skill, is suddenly way more valued than being a highly experienced Ruby-on-Rails programmer.

G. Software engineering process

The processes would adapt as a consequence of which organizational entities develop software and how their daily practice can be framed best in a systematic approach to support good communication and leads to high quality software that meets the needs. However, they are likely to include elements from open source, agile and user-centered approaches. Two possible scenarios: 1) we go towards a strictly formal and structured software engineering process (as in the aerospace and systems engineering disciplines) – we measure twice, and then cut once. That way, developers are developing very diligent formal proofs of their code before actually programming, because the cost of computing suddenly has gone up again 2) renegade programmers take over the net or what remains of it in informal structures and organization and collaborate in ad-hoc networks where minimal structured documentation and specification is needed in order to enable maintenance and evolution.

IV. POSSIBLE FUTURE NEEDS

In terms of different collapse scenarios implying future needs, we offer a number of hypotheses and possible solution elements.

A. Hypotheses

When reflecting upon various collapse scenarios, we came to the following hypotheses for the future of software engineering:

a. Under the various scenarios for collapse, the demand for innovations dries up. Consequently, Moore's law won't hold anymore.
b. Right now SE seems to rely on the abundance of resources, inter alia, hardware, money, energy, economy, work power. In many of the future scenarios, however, there will be a paradigm shift, as we move towards more resilience.
c. There are future scenarios that SE is not prepared for on different severity levels. This also depends on the nature of the systems that we will need, not only the tasks that are performed by software engineers.

B. Possible solution elements

We suggest a number of possible solution elements that may help addressing the arising challenges in those scenarios:

1. A framework with an overview of the scenarios and potential solution elements would help to propose future research agendas.
2. Some concepts from collaborative work and crowd sourcing might offer additional help as they are used in environments that are not always "reliable" (in terms of guarantees of contributions).
3. Further solution elements might be ad-hoc networks, modular software development, and distributed back-up.
4. Analogies in other research fields: We can draw parallels to the CHI and UBICOMP communities and learn from there.

V. CONCLUSION

Civilizational collapse may seem like a "fantastical scenario", and in fact has been called such by a reviewer of an NSF proposal submitted by several of the authors of this paper. However, there are global transformations under way that are difficult to avoid, if one looks at the world through scientific eyes. And, at least at present, there are not clear mechanisms in place in human communities to prevent those transformations from having profound impacts on the operation of industrial civilization. While collapse is not certain, it is one possible outcome of these transformations.

In the face of collapse, though, there will be opportunities for software systems to preserve human quality of life in the face of declining resource availability. And these software systems would mostly need to be built *during collapse*. Therefore, it seems appropriate now, in a time of great abundance, to devote some attention to a consideration of collapse as an insurance policy, and given our community's expertise, to the processes by which collapse-relevant software systems could be built. While the authors of this paper hope that this research never becomes relevant to put into practice, the growing possibility that it could become relevant suggests that now would be an excellent time to begin this discussion.

Acknowledgement: We would like to thank André van der Hoek and Marcel Pufal for discussing these ideas with us. So long, and thanks for all the fish.

VI. REFERENCES

[1] Penzenstadler, B. Infusing Green: Requirements Engineering for Green in and through software systems. 3rd Intl. Workshop on Req. Eng. for Sustainable Systems, 2014

[2] B. Penzenstadler, A. Raturi, D.J. Richardson, B. Tomlinson. Safety, Security, now Sustainability: the Non-Functional Requirement for the 21st Century. IEEE Software: Green Software, 2014.

[3] Silberman, M. S. Information systems for the age of consequences. *LIMITS '15*, submitted.

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

[5] http://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm

[6] http://www.dhs.gov/critical-infrastructure-sectors

[7] http://www.nsf.gov/pubs/2014/nsf14524/nsf14524.htm