De-escalating software: counterproductivity versus conviviality

Stephen Kell

29th April 2019

"[At first] the desirable effects of new scientific discoveries were easily measured and verified. [Later,] computer science still claimed continued progress, as measured by the new landmarks technologists set for themselves [... but] the total social cost exacted by this technology ceased to be measurable in conventional terms. Society can have no quantitative standards by which to add up the negative value of illusion, social control, drudgery, isolation, impotence and frustration produced by all-pervading software. The characteristic reaction to the growing frustration was further technological and bureaucratic escalation. ... It has become fashionable to say that where computer science and technology have created problems, it is only more scientific understanding and better technology that can carry us past them. The cure for bad management is more management. The cure for traffic jams is more roads. The attempt to overwhelm present problems by the introduction of more science is the ultimate attempt to solve a crisis by escalation."

Software is an escalating phenomenon, in at least two ways. Firstly, the complexity of even the simplest software deployments gets larger year on year [Wirth, 1995, Jones, 2006, Holzmann, 2009, Hatton et al., 2017]. Feature-equivalent software gains complexity in ways that cannot clearly be explained by niceties, or even directly by programmer laziness. That complexity is both in the machine resources required, and in the amount of code needed to realise a given application. These increases have consequences: for the overall tractability of working with that code, and for the capital costs of deploying and executing it. Secondly, software 'escalates' in that the best response of mainstream research and development has, at present, been to generate more of the same. Although researchers can point to appearances of forward progress on specific technical issues, this comes against a background of subtle and rising hidden costs—undermining and perhaps even negating, in cost/benefit terms, the value apparently created. Software engineers would recognise such a situation as a society-wide instance of the 'tar pit' of Brooks [1975].

If such a pattern is occurring, it would hardly be for the first time in the industrialised world. Despite the remarkable transformations that the past two-and-a-half centuries, we live in an era where it is not controversial to say that industrial thinking has exceeded its limits in many domains.

Thus far, software has mostly been seen in industrial terms. A well-known case is the talk of McIlroy [1969]. entitled 'Mass-Produced Software Components', given to the NATO Software Engineering Conference in 1968. By 'mass-produced', he meant that individual routines could be classified by both their abstract function and also other dimensions of variability (he mentions precision, robustness, generality, time-space performance and interface style), supposing that a 'catalogue' of code could be grown to cover a large number of points on each of these dimensions. The talk finished by noting 'what I have just asked for is simply industrialism, with programming terms substituted for some of the more mechanically oriented terms appropriate to mass production'. Despite the technical naïvety of this vision in hindsight [Kell, 2012], an industrial mindset has continued to prevail in software. Day-to-day activities and habits of increasingly many people are founded on an infrastructure of huge and growing complexity—which gives, but also takes away. As society as a whole exhibits increasing dependency on high-cost equipment and trained professional staffs, so power is concentrated in the hands of capital-holders and certified professionals at the expense of a majority who are neither.

Just as McIlroy substituted programming terms into the template of industrialism, so has our opening quotation into the words of Ivan Illich, from his work 'Energy and Equity' [Illich, 1974]: I have substituted programming terms for Illich's 'energy' and similar. He emphasises how the appearance of progress must be offset the background costs brought on by the industrial means. Illich's key recurring theme is the counterproductivity of institutions; he most notably critiqued road transport, modern education systems, and the role of medicine in the developed world of the late 20th century. In each case, the individuals engaged in perpetuating the institution may be acting rationally, but the outcome is madness.

This talk will look at software through an Illichian lens, exploring the ways in which the technical and social mechanisms of software conspire to create a similar dynamic. Specifically, I will consider several 'escalating' phenomena, such as: compiler optimisation and its systemic effects on code complexity and demand for further optimisation; the closed-box 'appliance' model of software and its consequent demand for more numerous appliances; the recurring phenomena of overprovisioning and misestimation and their consequent inflationary pressure on both hardware requirements and software complexity. In a selection of such cases, I will examine to what extent the Illichian idea of 'responsible self-limitation' [Illich, 1973] offer a viable means of de-escalation.

References

F. P. Brooks. The tar pit. In *The Mythical Man-Month: Essays on Software Engineering*. Addison Wesley, 1975.

L. Hatton, D. Spinellis, and M. van Genuchten. The long-term growth rate of evolving software: Empirical results and implications. *Journal of Software: Evolution and Process*, 29(5), 2017. doi: 10.1002/smr.1847. URL https://doi.org/10.1002/smr.1847.

G. Holzmann. OOPSLA keynote: Scrub and spin: Stealth use of formal methods in software development. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-768-4. doi: 10.1145/1639950.1705499. URL http://doi.acm.org/10.1145/1639950.1705499.

I. Illich. *Tools for Conviviality*. World Perspectives. Harper & Row, 1973.

I. Illich. *Energy and equity*. Ideas in progress. Harper & Row, 1974.

D. Jones. Why userspace sucks—or 101 really dumb things your app shouldn't do. In *Proceedings of the Linux Symposium, Volume One*, 2006.

S. Kell. *Black-box composition of mismatched software components*. PhD thesis, University of Cambridge, UK, 2012. URL http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.610557.

M. McIlroy. Mass-produced software components. In *Proceedings of NATO Conference on Software Engineering*, pages 88–98, 1969.

N. Wirth. A plea for lean software. *Computer*, 28(2):64–68, Feb. 1995. ISSN 0018-9162. doi: 10.1109/2.348001. URL https://doi.org/10.1109/2.348001.