Software Crisis 2.0

Brian Fitzgerald
Lero – the Irish Software Engineering Research Centre, University of Limerick

Individual efforts to improve software development capability are disjointed and not likely to deliver the capacity needed to keep pace with advances in hardware technology and the opportunities afforded by big data.

Although barely 50 years old, the software domain has already endured one well-documented crisis. Software Crisis 1.0 first arose in the 1960s, with software taking longer and costing more to develop than estimated, and not working very well when eventually delivered. Nevertheless, software is one of the computing revolution's big success stories, spurring a huge shift in how we go about our daily lives.

The past 50 years have also seen enormous advances in hardware capability, with dramatic reductions in hardware costs allied to equally impressive increases in processing power and device proliferation. An almost infinite amount of data is now available through ubiquitous sensors and applications such as Google. Complementing these "push" factors is a significant "pull" factor arising with the emergence of digital natives—users who have never known life without technology, but who have an enormous appetite for new technological applications. The advances in hardware technology along with the vast amounts of potentially available data afford truly enormous opportunities for individuals, business, and society.

Unfortunately, we haven't seen similar advances in relation to our software development capability, giving rise to what I call Software Crisis 2.0. Individual efforts seek to address this crisis—data analytics, parallel processing, new development methods, cloud services—but they're disjointed and not likely to deliver the software development capacity needed.

SOFTWARE CRISIS 1.0

The term *software* was first coined in 1958 (www.maa.org/mathland/mathtrek_7_31_00.html), but within 10 years, problems in software's development and delivery led to the phrase *software crisis* (P. Naur and B. Randell, eds., "Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee," NATO, 1968).

Over the years, several studies have confirmed Software Crisis 1.0. Per Flaatten and colleagues estimated the average project's development time at 18 months (*Foundations of Business Systems*, Dryden Press, 1989)—a conservative figure, given that other estimates put the figure at three years ("The Software Trap: Automate—or Else," *Business Week*, 9 May 1988, pp. 142-154) and even up to five years (T. Taylor and T. Standish, "Initial Thoughts on Rapid Prototyping Techniques," *ACM SIGSOFT Software Eng. Notes*, vol. 7, no. 5, 1982, pp. 160-166).

Perhaps this isn't surprising, given that an IBM study estimated 68 percent of all software projects overran their schedules (P. Bowen, "Rapid Application Development: Concepts and Principles," IBM document no. 94283UKT0829, 1994). In relation to

0018-9162/12/$31.00 © 2012 IEEE Published by the IEEE Computer Society

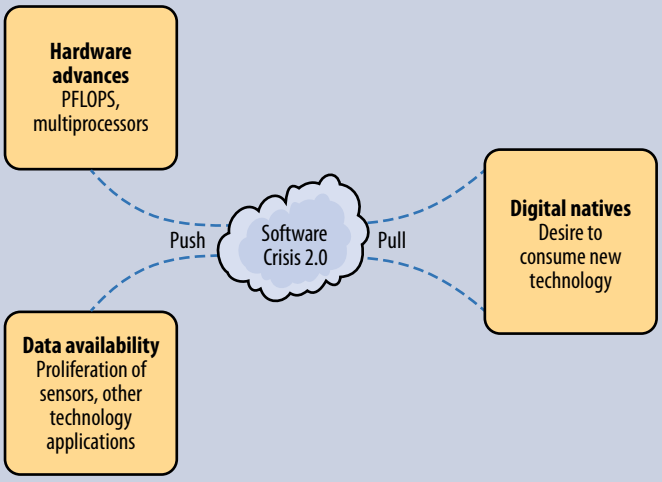

Figure 1. Software Crisis 2.0. The demand for data from digital natives, coupled with the huge volume of data now generated through ubiquitous mobile devices, sensors, and applications, has led to a new software crisis.

cost, the IBM study also suggested that development projects were as much as 65 percent over budget. Indeed, the term shelfware was coined to refer to software systems that are delivered but never used.

Although the Standish Group continues to paint a rather bleak picture of the high rates of software project failure (http://standishgroup.com/newsroom/chaos_manifesto_2011.php), its findings and methodology have been challenged (J. Eveleeens and C. Verhoef, "The Rise and Fall of the CHAOS Reports," *IEEE Software*, 2010, pp. 30-36). In fact, I believe that Software Crisis 1.0 has passed, and that the myriad, incremental advances pushing software development forward have changed our lives for the better. Changes in the practice have ultimately led the field to the point where software is now routinely developed largely on time, within budget, and within user expectation.

Unfortunately, Software Crisis 2.0 is now facing us. As Figure 1 shows, this crisis stems from the inability to produce software that can leverage the staggering increase in data generated in the past 50 years and the demands of the devices and users that can manipulate it.

BIG DATA AND THE RISE OF THE DIGITAL NATIVE

Many eye-catching figures and statistics illustrate the enormous advances that hardware capacity has experienced over the past half-century. Moore's law, for example, predicted the doubling of hardware capacity roughly every 18 months. To put that in perspective, if you had invested just a single dollar when Moore first made his prediction, and if the return on investment had kept pace accordingly, your net worth would be more than a $1 quadrillion, or $1 million billion.

Similar "laws" parallel Moore's prediction in relation to storage (Kryder's law) and network capacity (Butter's law). On each occasion when hardware appears to have halted due to an insurmountable challenge in the fundamental laws of physics—the impurity of atoms, light-wavelength limits, heat generation, radiation-induced forgetfulness, for example—new advances have emerged to overcome them.

Although it's extremely difficult to quantify the increased volume of electronic data that potentially exists, it undoubtedly follows a similar pattern. In 2005, Eric Schmidt, Google's CEO, suggested that the amount of data available electronically comprised 5 million Tbytes (5 million billion Mbytes), of which only .004 percent were indexed by Google (http://voices.yahoo.com/how-big-internet-does-one-measure-a-3285174.html). At that time, Schmidt estimated the amount of data to be doubling every five years.

In 2010, Dave Evans, Chief Futurist at Cisco Systems, estimated that 35 billion devices were connected to the Internet, which comes out to more than five times the planet's population (www.readwriteweb.com/archives/cisco_futurist_predicts_internet_of_things_1000_co.php). This figure is estimated to increase to 100 billion devices by 2020, giving rise to the concept of the Internet of Things (IoT)—the representation of uniquely identifiable things in an Internet-like structure. An exemplar project designed for the IoT is Hewlett-Packard's plan to place one trillion smart dust sensors all over the world as part of a planet-wide sensing network infrastructure. These sensors will detect a wide variety of factors, including motion, vibration, light, temperature, barometric pressure, airflow, and humidity, and will have obvious applications in transportation, health, energy management, and building automation.

Seeking to extend the IoT concept, Usman Haque proposes an "ecosystem of environments" through his Pachube project (www.haque.co.uk/pachube.php), a service that allows consumers tag and share real-time sensor data from objects and environments globally. The potential success of user-led innovation, cocreation of value, and high-profile crowdsourcing successes in solving complex R&D problems for NASA, Eli Lilly, and Du Pont highlight the crucial role of the digital native. By age 20, digital natives will have spent 20,000 hours online and can cope with, and even welcome, an abundance of information (S. Vodanovich, D. Sundaram,

and M. Myers, "Digital Natives and Ubiquitous Information Systems," Information Systems Research, vol. 21, no. 4, 2010, pp. 711-723). Rather than resisting technology, digital natives have an insatiable appetite for its new applications. By early 2012, mobile handset cellular subscriptions reached almost 6 billion (http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats). Although obviously not evenly distributed, this equates to almost 90 percent of the world's population.

Various initiatives have sought to address Software Crisis 2.0. Early efforts in computer-aided software engineering (CASE) sought to automate software development, but they haven't solved the problem. Similarly, initiatives in software architectures, patterns, reuse, and software product lines sought to provide improvements by building on existing foundations. Software capability maturity models and software development method-related initiatives such as method engineering have likewise been the subject of research.

Several initiatives exist to support hardware advances in the areas of multicore processing and parallel computing. More recently, autonomous computing initiatives have sought to deliver self-maintaining systems that would evolve automatically, an attempt that builds on previous artificial intelligence initiatives such as genetic algorithms. Efforts in relation to the Semantic Web and ontologies have also sought to address "big data" challenges.

Given the scarcely comprehensible increases in hardware power and data capacity to date, it's perhaps surprising that we have yet to find a silver bullet that delivers even a modest one-order-of-magnitude improvement in software productivity. Can we even imagine what life would be like if the software area had evolved at the same pace as hardware and data? Wirth's law effectively summarizes the comparative evolution in the software domain—namely, that software is getting slower more rapidly than hardware becomes faster (N. Wirth, "A Plea for Lean Software," *Computer*, vol. 28, no. 2, 1995, pp. 64-68).

We've entered an era in which the limits of our imagination should be the only limiting factor in taking advantage of the past 50 years' worth of advances to help solve intractable problems in areas such as healthcare, energy efficiency, and climate control. But the first step in solving any problem is to acknowledge its actual existence. This article might seem controversial, but it seeks to accomplish the easy part—naming the problem. The much more complex subsequent steps require the identification of an agenda to resolve it.

Acknowledgements

I acknowledge feedback from Michael Myers and financial support from Science Foundation Ireland grant 10/CE/I1855 to Lero - the Irish Software Engineering Research Centre.

Brian Fitzgerald *holds the Frederick Krehbiel Chair in Innovation in Global Business and Technology at the University of Limerick. Contact him at bf@ul.ie.*

Editor: Mike Hinchey, Lero—The Irish Software Engineering Research Centre; mike.hinchey@lero.ie

Selected CS articles and columns are available for free at http://ComputingNow.computer.org.