Software Crisis 2.0

Abstract

Although only about 50 years old, the software domain has already endured one well documented crisis, which was identified early in its evolution in the 1960s. Simply summarised, the initial software crisis – Software Crisis 1.0 as it is termed here – referred to the fact that software took longer to develop than estimated, cost more to develop than estimated, and did not work very well when eventually delivered. Over the years many studies have confirmed the Software Crisis 1.0. Nevertheless, software has been one of the key success stories of the past 50 years and has truly revolutionised modern life. However, over the past 50 years, there have also been enormous advances in hardware capability – dramatic reductions in hardware costs allied to dramatic increases in processing power and proliferation of devices; almost infinite amounts of data are now available through ubiquitous sensors and through applications such as Google. Complementing these 'push' factors, there is a significant 'pull' factor arising through the emergence of 'digital native' technology consumers who have never known life without technology. The opportunities for individuals, business and society afforded by the advances in hardware technology and the vast amounts of data potentially available, when allied to the insatiable appetite of digital natives, are truly enormous. Unfortunately there have not been similar advances in relation to our software development capability, and thus the critical limiting factor in realising the potential of the advances mentioned above is again software – the Software Crisis 2.0 as I label it here. Individual efforts are seeking to address this crisis – data analytics, parallel processing, new development methods, cloud services – but these are disjointed, and not likely to deliver the software development capacity needed.

Software Crisis 1.0

The term 'software' was coined in 1958 (Peterson 2000), but within ten years, problems in the development of software led to the coining of the phrase 'software crisis' (Naur and Randall 1968). The software crisis referred to the realisation that software took longer to develop than estimated, cost more to develop than estimated, and did not work very well when eventually delivered. Over the years, several studies have confirmed these tripartite aspects of the software crisis. For example, in relation to development time-scales: Flaatten *et al*. (1989) estimated development time for the average project to be about 18 months — a conservative figure perhaps given that other estimates put the figure at about three years (Business week 1988) or even up to five years (Taylor and Standish 1982). Also, an IBM study estimated that 68 per cent of projects overran schedules (Bowen 1994). In relation to cost, the IBM study suggested that development projects were as much as 65 per cent over budget (Bowen 1994), while a Price Waterhouse study in the UK in 1988 concluded that £500m was being lost per year through ineffective development. Furthermore, in relation to performance, the IBM study found that 88 per cent of systems had to be radically redesigned following implementation (Bowen 1994). Similarly, a UK study found that 75 per cent of systems delivered failed to meet users expectations. This has led to the coining of the term 'shelfware' to refer to those systems which are delivered but never used.

However, the initial software crisis has largely been resolved. While the Standish Chaos Report continues to paint a rather bleak picture of high rates of software project failure – estimated at 68%, for example (Standish Group 2009), the Chaos report findings and methodology have been challenged (e.g. Eveleens and Verhoef 2010, Glass, 2006). Although there has been no 'silver bullet' advance, using Brooks (1987) term, which affords an order of magnitude improvement in software development productivity, a myriad of advances have been made in more incremental ways, and software is now routinely developed largely on time, within budget, and works well. Software is really the success story of modern life. Everything we do, how we work, travel, communicate, entertain ourselves has been dramatically altered and enhanced by the capabilities provided by software.

However, a new software crisis is now upon us, one which I term 'Software Crisis 2.0'. Basically this arises as a result of the inability to produce software to leverage the absolutely staggering increase in the volume of data being generated, in turn allied to the enormous amount of computational power offered by the many hardware devices also available, and both complemented by the appetite of the newly emerged 'digital native' consumer (see Fig 1).

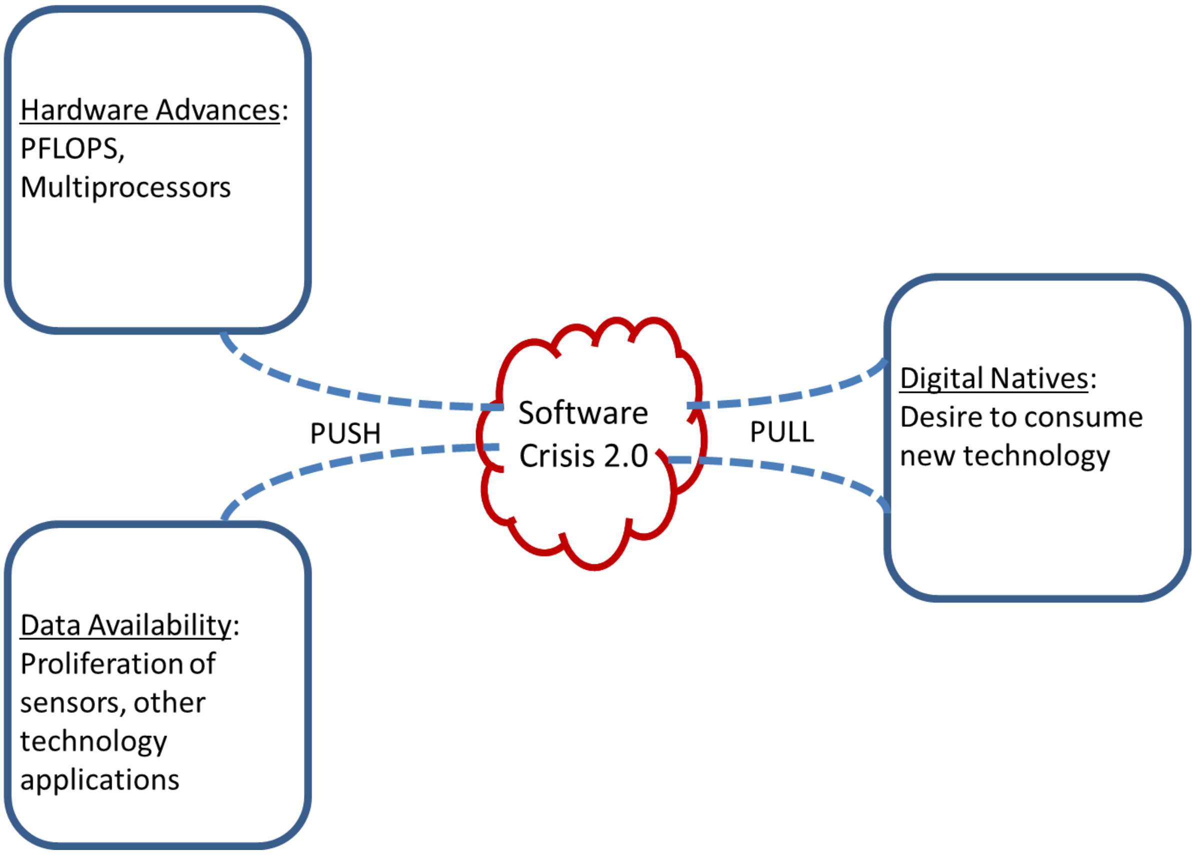
Hardware Advances: PFLOPS, Multiprocessors
Data Availability: Proliferation of sensors, other technology applications
PUSH
Software Crisis 2.0
PULL
Digital Natives: Desire to consume new technology

Fig 1 Software Crisis 2.0

The Evolution of Hardware

There are many eye-catching figures and statistics that illustrate the enormous advances in the evolution of hardware capacity over the past half-century or so. Moore's Law, for example, predicted the doubling of hardware capacity roughly every 18 months or so. To put that in perspective, if one had invested just a single dollar when Moore was declaring his prediction initially, and if return on investment had kept pace accordingly, the individual's net worth would be over a quadrillion dollars – that is over $1,000,000,000,000,000 or one million billion dollars. Moore's law is paralleled by similar 'laws' in relation to storage capacity (Kryder's Law) and network capacity (Butter's Law) which portray similar exponential predictions.

On each occasion when hardware appears to be halted due to an insurmountable challenge in the fundamental laws of physics – the impurity of atoms, sculpting light-wavelength limits, heat generation, radiation-induced forgetfulness, for example – new advances have emerged to overcome these problems as we move into the quantum computing area.

Volume of Data

While it is extremely difficult to quantify the increases in the volume of electronic data that potentially exists, there is undoubtedly a similar pattern of exponential increases paralleling that of the hardware arena. Eric Schmidt, CEO of Google, suggested in 2005 that the amount of data available electronically comprised 5 million terabytes (that is 5 million billion megabytes), of which only .004% was being indexed by Google (Schmidt 2005). He estimated the amount of data as doubling every five years.

Dave Evans, Chief Futurist at Cisco Systems estimated in 2010 that there were about 35 billion devices connected to the Internet, which is more than five times the population of the planet (Jeffries 2010). This figure is estimated to increase to 100 billion devices by 2020. This has given rise to the concept of the "Internet of Things" (IoT) (Ashton 2009). An exemplar project designed for the IoT is the plan by HP as part of the Central Nervous System for the Planet (CeNSE) project to place a trillion 'smart dust' sensors all over the planet as a planet-wide sensing network infrastructure. These sensors can detect a wide variety of factors, including motion, vibrations, light, temperature, barometric pressure, airflow and humidity, and have obvious applications in transportation, health, energy management and building automation.

To cope with this proliferation of devices, a migration is planned from the IPv4 protocol which has about four billion unique addresses to the IPv6 protocol which can support 21^{28} addresses – enough to uniquely address every grain of sand on every beach in the world. In this brave new world, the vast majority of communications will be machine-to-machine rather than machine-to-person, thereby generating an enormous amount of electronic information which is available for processing.

The Emergence of 'Digital Natives'

An interesting distinction has been drawn between ‘digital immigrants’ – those who began using digital technology at some stage during their adult lives, and ‘digital natives’ – those who have been immersed in the world of technology since birth and have as a consequence developed a natural fluency for technology (Prensky 2001). By the age of 20, digital natives will have spent 20,000 hours online (Valkenburg and Peter 2008) and can cope with, and indeed even welcome, an abundance of information (Vodanovich et al 2010). This category of digital native consumer represents a significant ‘pull’ factor in seeking to take advantage of the opportunities afforded by advances in processing power and increased availability of data.

A survey by the UN Body, International Telecommunications Union (ITU), reported that about a quarter of the world's 7 billion people use the Internet. Also, mobile phone subscriptions have been numbered at over four billion. There is not a simple equivalence of subscription to individual user – in Europe mobile subscription exceeds population by 11 per cent, whereas in Africa, mobile phones may be shared – but clearly this is a huge number of consumers. Also, the dramatic increase in mobile penetration suggests that consumers in the developing world are leap-frogging landline technology to the smart mobile devices.

Seeking to extend the Internet of Things concept, Usman Haque proposes an “ecosystem of environments” though his Pachube project, a service that lets consumers tag and share real-time sensor data from objects and environments globally. The success of user-led innovation, co-creation of value, and high profile crowdsourcing successes in solving complex R&D problems for NASA, Eli Lilly and Du Pont provides real testimony to the potential of the digital native

Software Crisis 2.0 – the bottleneck

Various initiatives have sought to address aspects of this problem. Early efforts in computer-aided software engineering (CASE) sought to automate the software development activity, but this has not solved the problem. Similarly, early initiatives in software architectures, software patterns and software product lines sought to provide improvements by building on past learning. In the area of software process, maturity models and software development method–related initiatives such as method engineering have been the subject of research. In relation to leveraging hardware advances, there have been several initiatives in the area of multi-core processing and parallel computing. More recently, autonomous computing initiatives have sought to deliver self-maintaining systems which would evolve automatically. In relation to the so called ‘big data’, the field of data analytics is emerging. This builds on previous initiatives in artificial intelligence such as genetic algorithms. Efforts in relation to the semantic web and ontologies have also sought to address ‘big data’ challenges. The manner in which technology pervades all our lives has also given rise to complex challenges in the information security area. However this is just a brief snapshot of initiatives that seek to address the above challenges. The main point to note is that these initiatives are all fragmented and disjoint when viewed from the perspective of the software crisis 2.0.

Given the scarcely comprehensible increases in hardware power and data capacity mentioned above, it is perhaps surprising that there has not been a ‘silver bullet’ to deliver even a modest one order of magnitude improvement in software productivity. Without wishing to deny the enormous advances that have been brought about by software, which has truly revolutionised life and society in the 20th and 21st centuries, it is intriguing to imagine what life would be like if the software area had evolved at the same pace as that of hardware and data. But that has not been the case: Wirth’s Law (Wirth 1995) effectively summarises the comparative evolution in the software domain, namely that software is getting slower more rapidly than hardware becomes faster.

Given the advances outlined above, we have entered an era when the limits of our imagination should be the only limiting factor in taking advantage of the advances outlined above to help solve intractable problems and deliver in areas such health care, energy efficiency, climate control, entertainment and the like. The first step in solving a problem is to acknowledge its actual existence. This article may seem controversial, but it seeks to accomplish the easy first part. The much more complex subsequent steps require the identification of an agenda to resolve the problem.

References

Kevin Ashton: That 'Internet of Things' Thing. In: *RFID Journal*, 22 July 2009., .

Eveleeens, J and Verhoef, C (2010) The Rise and Fall of the Chaos reports, *IEEE Software*, pp.30-36.

R. Glass, "The Standish Report: Does It Really Describe a Software Crisis?" *Comm. ACM*, vol. 49, no. 8, 2006, pp. 15–16.

Peterson, I (2000) Software's Origin, http://www.maa.org/mathland/mathtrek_7_31_00.html (accessed Oct 2011)

Naur, P., and Randell, B. (eds.) (1968) *Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee*. Brussels: Scientific Affairs Division, NATO.

Flaatten, P., McCubbrey, D., O'Riordan, P. and Burgess, K. (1989) *Foundations of Business Systems*, Dryden Press, Chicago.

Business Week (1988) The software trap: automate—or else, *Business Week*, May 9, 142-154.

Taylor, T, and Standish, T. (1982) Initial thoughts on rapid prototyping techniques, *ACM SIGSOFT Software Engineering Notes*, **7**, 5, 160-166.

Bowen, P. (1994) Rapid Application Development: Concepts and Principles, IBM Document No. 94283UKT0829.

Standish Group. 2009. *The CHAOS Report*, The Standish Group, Boston, MA.

Niklaus Wirth (February 1995). "A Plea for Lean Software". *Computer* 28 (2): pp. 64–68. doi:10.1109/2.348001

Adrianne Jeffries / July 29, 2010 A Sensor In Every Chicken: Cisco Bets on the Internet of Things,http://www.readwriteweb.com/archives/cisco_futurist_predicts_internet_of_things_1000_co.php

Rafi Haladjian, inventor of the communicating rabbit Nabaztag, 25 May 2009 in Santucci