Part I

The Software Landscape

1

Software Crisis 2.0

Brian Fitzgerald

Lero – The Irish Software Research Centre, University of Limerick, Limerick, Ireland

1.1 Software Crisis 1.0

> In 1957, the eminent computer scientist, Edsger Dijkstra, sought to record his profession as "Computer Programmer" on his marriage certificate. The Dutch authorities, probably more progressive than most, refused on the grounds that there was no such profession. Ironically, just a decade later, the term "software crisis" had been coined, as delegates at an international conference in 1968 reported a common set of problems, namely that software took too long to develop, cost too much to develop, and the software which was eventually delivered did not meet user expectations.

In the early years of computing during the 1940s, the computer was primarily used for scientific problem solving. A computer was needed principally because of its speed of mathematical capability, useful in areas such as the calculation of missile trajectories, aerodynamics, and seismic data analysis. The users of computers at the time were typically scientific researchers with a strong mathematical or engineering background who developed their own programs to address the particular areas in which they were carrying out research. For example, one of the early computers, ENIAC (Electronic Numerical Integrator and Calculator), which became operational in 1945, by the time it was taken out of service in 1955 "had probably done more arithmetic than had been done by the whole human race prior to 1945" [1].

During the 1950s, the use of computers began to spread beyond that of scientific problem solving to address the area of business data processing [2]. These early data processing applications were concerned with the complete and accurate capture of the organization's business transactions, and with

Software Technology: 10 Years of Innovation in IEEE Computer, First Edition.
Edited by Mike Hinchey.

automating routine clerical tasks to make them quicker and more accurate. This trend quickly spread, and by 1960, the business data processing use of computers had overtaken the scientific one [3]. Once underway, the business use of computers accelerated at an extremely rapid rate. The extent of this rapid expansion is evidenced by the fact that in the United States, the number of computer installations increased more than twentyfold between 1960 and 1970 [4].

However, this rapid expansion did not occur without accompanying problems. The nature of business data processing was very different from the computation-intensive nature of scientific applications. Business applications involved high volumes of input and output, but the input and output peripherals at the time were very slow and inefficient. Also, memory capacity was very limited, and this led to the widespread conviction among developers that good programs were efficient programs, rather than clear, well-documented, and easily understood programs [3]. Given these problems, writing programs required much creativity and resourcefulness on the part of the programmer. Indeed, it was recognized that it was a major achievement to get a program to run at all in the early 1960s [5].

Also, there was no formal training for developers. Programming skills could only be learned through experience. Some programmers were drawn from academic and scientific environments and thus had some prior experience. However, many programmers converted from a diverse range of departments. As Friedman [3] describes it

> People were drawn from very many areas of the organization into the DP department, and many regarded it as an 'unlikely' accident that they became involved with computers.

Also, during the 1960s, the computer began to be applied to more complex and less-routinized business areas. Aron [6] identifies a paradox in that as the early programmers improved their skills, there was a corresponding increase in the complexity of the problem areas for which programs had to be written.

Thus, while the term "software" was only introduced in 1958 [7], within 10 years, problems in the development of software led to the coining of the phrase "software crisis" at the NATO Conference in Garmisch [8]. The software crisis referred to the reality that software took longer to develop and cost more than estimated, and did not work very well when eventually delivered.

Over the years, several studies have confirmed these three aspects of the software crisis. For example, in relation to development timescales: Flaatten et al. [9] 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 3 years [10] or even up to 5 years [11]. Also, an IBM study estimated that 68% of projects overran schedules [12]. In relation to the cost, the

IBM study suggested that development projects were as much as 65% over budget [12], while a Price Waterhouse study in the United Kingdom in 1988 concluded that £500 million was being lost per year through ineffective development. Furthermore, in relation to performance, the IBM study found that 88% of systems had to be radically redesigned following implementation [12]. Similarly, a UK study found that 75% of systems delivered failed to meet users expectations. This has led to the coining of the term "shelfware" to refer to those systems that are delivered but never used.

Notwithstanding the bleak picture painted above, the initial software crisis has largely been resolved, while the Standish Chaos Report continues to report high rates of software project failure – estimated at 68%, for example [13].[1] Although there has been no "silver bullet" advance, using Brooks [16] 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 delivered on time, within budget, and meets user requirements 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.

1.2 Software Crisis 2.0

However, a new software crisis is now upon us, one that I term "Software Crisis 2.0." Software Crisis 2.0 is fuelled by a number of "push factors" and "pull factors." Push factors include advances in hardware such as that perennially afforded by Moore's law, multiprocessor and parallel computing, big memory servers, IBM's Watson platform, and quantum computing. Also, concepts such as the Internet of Things and Systems of Systems have led to unimaginable amounts of raw data that fuel the field of data analytics. Pull factors include the insatiable appetite of digital native consumers – those who have never known life without computer technology – for new applications to deliver initiatives such as the quantified self, lifelogging, and wearable computing. Also, the increasing role of software is evident in the concept of software-defined * (where * can refer to networking, infrastructure, data center, enterprise). The Software Crisis 2.0 bottleneck arises from the inability to produce the volume of software necessary to leverage the absolutely staggering increase in the volume of data being generated, which 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 and a

1 It is worth noting that the Chaos report findings and methodology have been challenged (e.g., [14,15]).

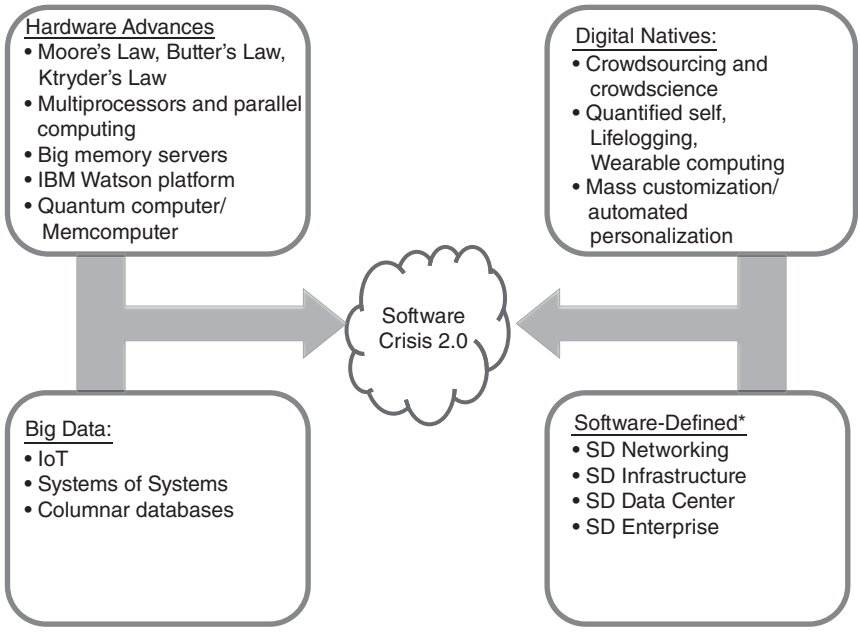

Figure 1.1 Software Crisis 2.0.

world where increasingly software is increasingly the key enabler (see Figure 1.1).

1.2.1 Hardware Advances

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 illustrate this in a more familiar context, if one had invested just a single dollar in some shares when Moore was declaring his prediction initially in 1964, and if the stock market return on this shares had kept pace accordingly with Moore's prediction, the individual's net worth would now exceed $17 billion – not bad for a $1 investment. 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.

Moore's law is paralleled by similar "laws" in relation to storage capacity (Kryder's law) and network capacity (Butter's law) that portray similar

exponential performance with decreasing costs. Big memory servers are disrupting the technology landscape with servers now capable of providing terabytes of physical memory. This has led to the observation that disks have become the new magnetic tape, as it is now possible to use physical memory for random access operations and to reserve the traditional random access disk for purely sequential operations.

1.2.1.1 Parallel Processing

In the era of Software Crisis 1.0, the programming paradigm was one based on serial computation. Instructions were executed one at a time on a single processor. Parallel processing allows programs to be decomposed to run on multiple processors simultaneously. Two significant advantages of parallel processing are significantly faster execution times and lower power consumption. Certain types of processing graphics, cryptography, and signal processing, for example, are suited to parallel decomposition.

1.2.1.2 IBM Watson Technology Platform

The IBM Watson platform uses a natural language and machine learning to allow a computer to appreciate context in human language, a task in which computers traditionally perform very poorly. Watson achieved widespread recognition for its ability to beat human experts in the game of Jeopardy. Configuring Watson to achieve this victory was far from trivial as it was underpinned by 200 million pages of content (including the full text of Wikipedia), 2800 processor cores, and 6 million logic rules [17]. Watson has been deployed in a variety of contexts: as a call center operator, hotel concierge, and chef as it has even published its own cookbook. However, IBM believes it has the power to revolutionize human–computer interaction with many applications in domains very beneficial to society, such as medicine where Watson technology is being used to create the ultimate physician's assistant as a cancer specialist. IBM estimate that a person can generate 1 million gigabytes of health-related data across his or her lifetime – roughly the equivalent of 300 million books. These large data problems are ones in which Watson can perform well as Watson can process 500GB (the equivalent of a million books) per second.

1.2.1.3 Quantum Computer and the Memcomputer

Although still primarily theoretical in nature, quantum computing has significant advantages over the traditional digital computer in terms of its vastly superior performance in solving certain problems. Digital computers are so called because they are based on the binary system of two digital states, 0 and 1, which conveniently map to electronic switching states of "off" or "on." Quantum computers, on the other hand, operate on quantum bits or *qubits* in short. Qubits are not restricted to being in a state of 0 or 1; rather they can be either 0

or 1 or through what is termed superposition, they can be both 0 and 1 (and all points in between) at the same time. As mentioned, digital computers typically perform only one instruction at a time, whereas a quantum computer performs many calculations simultaneously, and hence it is inherently delivering parallelism. Estimates suggest that a quantum computer could solve within seconds a problem that might take a digital computer 10,000 years to calculate [18]. Quantum computers are suited to optimization problems, an application of artificial intelligence that IBM Watson is also addressing, albeit still operating within the overall digital computer paradigm. In 2015, Google and NASA announced their collaboration on the D-Wave 2X quantum computer that has over 1000 qubits.[2]

Other alternatives to the traditional digital computer are also being investigated. These include the memcomputer, so called because it seeks to mimic the functioning of the memory cells of the human brain [19]. However, while the D-Wave 2X quantum computer requires an environment 150 times colder than interstellar space, in contrast, the memcomputer operates at room temperature. The fundamental innovation in the memcomputer is that, like the human brain, it stores and processes information in the same physical space, thereby overcoming a central problem in traditional digital computers that of transfer of information between the central processing unit and memory. The traditional computer uses millions of times more power than the brain on such data transfer, which is ultimately extremely wasteful of time and energy as such transfers do not add any essential value.

1.2.2 "Big 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 0.004% was being indexed by Google. 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 [20]. This figure was estimated to increase to 100 billion devices by 2020. This has given rise to the concept of the "Internet of Things" (IoT) [21] or network of everything. 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 would detect a wide

2 http://www.dwavesys.com/blog/2015/08/announcing-d-wave-2x-quantum-computer

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.

Similar predictions were made, such as that of the World Wireless Research Forum's (WWRF) that there would be 7 trillion devices for the world's 7 billion people by 2017 – which is a thousand devices for every human being – all of which would be intelligently connected to create an individual personal network for all. This suggests that more structure is needed for IoT, in that a "systems of systems" approach is necessary to govern and deliver these networks.

To cope with this proliferation of devices, a migration is underway from the IPv4 protocol that has about four billion unique addresses to the IPv6 protocol that 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 that is available for processing.

Big Data needs has several technological implications. For example, columnar databases that invert the traditional row-oriented relational databases can perform much more efficiently on search tasks as the data becomes the primary key effectively. Columnar databases lend themselves to greater compression and therefore require less space and can achieve faster transfer rates.

Complementing the basic "push" factors of hardware advances and big data availability are a number of "pull" or demand factors that underpin the need for more software. These include the software-hungry "digital natives" and the trend toward software-defined *, where * can represent networking, infrastructure, datacenter, or enterprise, reflecting the fact that software is the primary mechanism mediating the modern world. These pull factors are discussed next.

1.2.3 Digital Natives Lifelogging and the Quantified Self

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 [22]. By the age of 20, digital natives will have spent 20,000 h online [23] and can cope with, and indeed even welcome, an abundance of information [24]. 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. The advent of wearable computing fuels big data and has led to initiatives such as lifelogging and the quantified self. With such initiatives individuals can collect data about

all aspects of their daily lives – diet, health, recreation, mood states, performance – in some cases recording a terabyte of data per annum [25].

The paradoxical success of the open-source software phenomenon has led to a broader interest in crowd science or citizen science as a collaborative model of problem analysis and solving. Notable areas of success are user-led innovation, cocreation of value, and high-profile crowdsourcing of solutions for solving complex R&D problems in NASA, Eli Lilly, and Du Pont, which provides real testimony to the potential of the digital native.

Mass customization has been succinctly defined as "producing goods and services to meet individual customer's needs with near mass production efficiency" [26]. While not a new concept, it resonates well with catering to the personal needs of the digital native. Also, it is now typically delivered through some form of software-mediated configurability to meet individual customer needs. The concept of automated personalization is linked to the desired benefits of big data.

1.2.4 Software-Defined*

The increasing demand for software already discussed is fuelled by the increasing capability of software to perform tasks that were previously accomplished through hardware. This is evident in phenomena such as software-defined networking [27] or software-defined infrastructure [28], even software-defined datacenters [29], right through to the concept of the software-defined enterprise that has enough intelligence to automate all business processes. This is also evident in the move beyond IoT to Systems of Systems where the sensors and sources of data, such as household appliances, are fully integrated into web-enabled systems capable of utilizing machine-learning techniques to offer real-time data analytics on the morass of acquired raw data, with the ultimate goal of enabling societal benefits for citizens through the provision of useful and precisely customized information – the quantified self, for example.

1.3 Software Crisis 2.0: The Bottleneck

Given these "push" and "pull" factors, it is clear that a massive increase in the volume of software being produced is required to address these emerging initiatives. This creates a Software Crisis 2.0 bottleneck as we illustrate further. There are two dimensions to this crisis. One is the massive increase in the volume of software required to fuel the demand in new domains where software has not been always of primary significance – medicine and healthcare for example – where terabytes of raw data need to be analyzed to provide useful actionable insights. The second dimension, however, is a more challenging one as it requires software development practitioners to acquire fundamentally

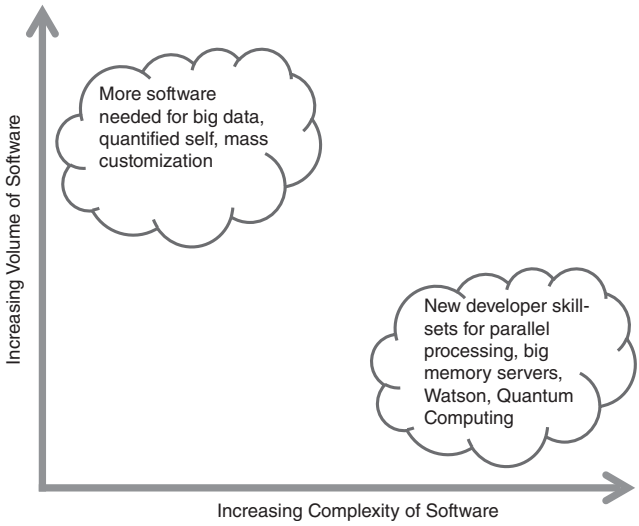

Figure 1.2 Increasing volume of software and complex developer skill sets.

different new skills to take advantage of advances in hardware – parallel processing, big memory servers, and quantum computing, for example – these will all require significant new skills on the part of practitioners (see Figure 1.2).

1.3.1 Significant Increase in Volume of Software Required

> "Our organization has become a software company. The problem is that our engineers haven't realized that yet!"

This is how the Vice President for Research of a major semiconductor manufacturing company, traditionally seen as the classic hardware company, characterized the context in which software solutions were replacing hardware in delivering his company's products. This situation is replicated across several business domains as the transformation to software has been taking place for quite some time. The telecommunications industry began the move to *softwareization* in the 1970s with the introduction of computerized switches, and currently, the mobile telephony market is heavily software focused. The automotive industry has very noticeably been moving toward softwareization since the 1960s–today, 80–90% of innovations in the automotive industry are enabled by software [30,31]. This is evidenced in the dramatic increase in the numbers of software engineers being employed in proportion to the numbers employed in traditional engineering roles. Indeed, an extremely striking example of the growing importance of software arises in the automotive industry. In 1978, a printout of the lines of code in the car would have made a paper stack

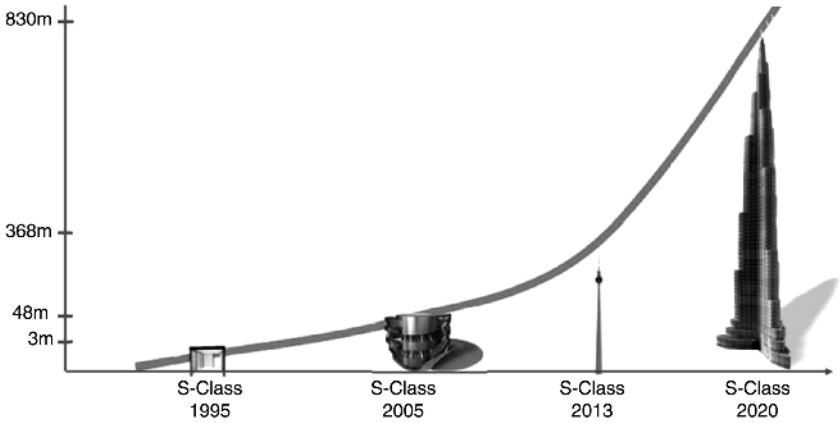

Figure 1.3 Height of software printout in Mercedes S-Class [32].

about 12 cm high. By 1995, this was already a 3-m-high stack, and by 2005, the printout was about 50 m tall. By 2013, the printout had grown to 150 m in height. By 2020, the estimate is that the stack would be a staggering 830 m tall, higher than the Burj Khalifa – the tallest man-made structure in the world [32]. This is illustrated graphically in Figure 1.3.

1.3.2 New Skill Sets Required for Software Developers

This demand for additional software is clearly replicated in several other industry domains. However, the software required for these domains is typically "more of the same" in that no major paradigm change is present that requires developers to possess new skills and techniques. In the overall backdrop to this software bottleneck, however, it is worth bearing in mind that estimates suggest the population of professional software engineers worldwide to comprise no more than 500,000 people [33]. Clearly, there are more software development practitioners in the world, and development resources may even be boosted by a general willingness for additional people to get involved based on a crowd-sourcing model. However, the skills required in this brave new world are not those possessed by the average software developer.

In the area of parallel processing on multicore architectures, for example, a number of fundamental challenges emerge. The traditional programming paradigm is one of runtime task allocation and scheduling, that is, the operating system allocates tasks to processors and takes care of scheduling and load balancing. In a multicore architecture, these decisions can be made at design-time or compile-time and developers need to design program threads accordingly. The analysis, design, and debug phases are significantly more challenging, and also an optimization/tuning phase is necessary. In the analysis phase, for

example, new questions arise. For example, not all code might benefit from parallelization. Code which is executed more frequently would be likely to lead to greater benefit, but code may be so simple that no performance benefit may arise through any potential parallelism, or there may not be any parallelizable loops. In the design phase, issues such as method of threading and decomposition need to be addressed. In the debug phase, handling data races and deadlocks and implementing thread synchronization accordingly is the focus. The optimization/tuning phase considers performance issues such as the amount of code parallelism, and whether performance benefits can be achieved as the number of processors increases.

1.3.2.1 From Runtime to Design-Time Switch

This is an interesting issue as much focus in recent times in software engineering has been on runtime adaptation, that is delaying decisions that are normally taken at design time until runtime [34]. This is evident in work on adaptive security and privacy, for example [35]. However, in the case of programming for multicore processors, issues such as the allocation of tasks to processors and load-balancing must be done at design time.

Programming big memory servers is also likely to lead to significant new programming challenges. The concept of garbage collection, for example, could be extremely problematic in a 64 terabyte single RAM space. Likewise, the mechanisms for dealing with a crash, or the notions of transient and persistent memory need to be reconceptualized when programming for a big memory server environment.

Quantum computing is not likely to replace traditional computing in the near future. However, understanding the quantum concepts of superposition and entanglement is far from trivial. At present, only a certain class of problem lends itself to be solved more efficiently by a quantum computer, optimization problems for example. Analyzing and understanding such problems is clearly not the *forte* of the majority of software developers at present. Quantum computing languages have also been created – QCL or quantum computing language [36] and Quipper [37], for example, but the quantum operations in these languages will be completely alien to the traditionally trained developer.

1.4 Conclusion

Given the scarcely comprehensible increases in hardware power and data capacity mentioned already, 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 revolutionized life and society in the twentieth and twenty-first 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 [38] effectively summarizes the comparative evolution in the software domain, namely, that software is getting slower more rapidly than the hardware is becoming faster.

References

1 B. Cohen (1988) The computer: a case study of the support by government, especially the military, of a new science and technology. Cited in Pickering, A. Cyborg history and WWII regime.
2 G. Davis and M. Olson (1985) *Management Information Systems: Conceptual Foundations, Structure and Development*, 2nd Edition, McGraw-Hill, New York.
3 A. Friedman (1989) *Computer Systems Development: History, Organisation and Implementation*, John Wiley & Sons, Ltd., Chichester.
4 C. Lecht (1977) *The Waves of Change*, McGraw-Hill, New York.
5 M. Shaw (1990) Prospects for an engineering discipline of software. *IEEE Software* **7**, 15–24.
6 J. Aron (1976) Estimating resources for large programming systems. In Naur, P. Randell, B., and Buxton, J. (eds.), *Software Engineering: Concepts and Techniques*, Charter Publishers, New York, 206–217.
7 I. Peterson (2000) Software's Origin. Available at http://www.maa.org/mathland/mathtrek_7_31_00.html (accessed Oct. 2011).
8 P. Naur, and B. Randell (eds.) (1968) *Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee*. Scientific Affairs Division, NATO, Brussels.
9 P. Flaatten, D. McCubbrey, P. O'Riordan, and K. Burgess (1989) *Foundations of Business Systems*, Dryden Press, Chicago.
10 Anonymous (1988) The software trap: automate—or else, Business Week, 142–154.
11 T. Taylor and T. Standish (1982) Initial thoughts on rapid prototyping techniques. *ACM SIGSOFT Software Engineering Notes*, **7** (5), 160–166.
12 P. Bowen (1994) Rapid Application Development: Concepts and Principles, IBM Document No. 94283UKT0829.
13 Standish Group (2009) The CHAOS Report, The Standish Group, Boston, MA.
14 J. Eveleeens and C. Verhoef (2010) The rise and fall of the Chaos reports. *IEEE Software*, **27**, 30–36.
15 R. Glass (2006) The Standish report: does it really describe a software crisis? *Communications of the ACM*, **49** (8), 15–16.
16 F. Brooks (1987) No silver bullet: essence and accidents of software engineering. *IEEE Computer Magazine April*, 10–19.

17 I. Paul (2010) IBM Watson Wins Jeopardy, Humans Rally Back, PCWorld. Available at http://www.pcworld.com/article/219900/IBM_Watson_Wins_Jeopardy_Humans_Rally_Back.html

18 D. Basulto (2015) Why Google's new quantum computer could launch an artificial intelligence arms race, Financial Review, Available at http://www.afr.com/technology/why-googles-new-quantum-computer-could-launch-an-artificial-intelligence-arms-race-20151228-glvr7s

19 F. Traversa, C. Ramella, F. Bonani, and M. Di Ventra (2015) Memcomputing *NP*-complete problems in polynomial time using polynomial resources and collective states. *Science Advances*, **1** (6). doi: 10.1126/sciadv.1500031.

20 A. Jeffries (2010) A Sensor In Every Chicken: Cisco Bets on the Internet of Things. Available at http://www.readwriteweb.com/archives/cisco_futurist_predicts_internet_of_things_1000_co.php

21 K. Ashton (2009) That 'Internet of Things' Thing. RFID Journal.

22 M. Prensky (2001) Digital natives, digital immigrants. *On Horizon* **9** (5), 1–2.

23 P. M. Valkenburg and J. Peter (2008) Adolescents' identity experiments on the Internet: consequences for social competence and self-concept unity. *Communication Research* **35** (2), 208–231.

24 S. Vodanovich, D. Sundaram, and M. Myers (2010) Digital natives and ubiquitous information systems. *Information Systems Research* **21** (4), 711–723.

25 C. Gurrin, A. Smeaton, and A. Doherty (2014) LifeLogging: Personal Big Data. doi: 10.1561/1500000033.

26 M.M. Tseng and J. Jiao (2001) Mass Customization *Handbook of Industrial Engineering, Technology and Operation Management*, 3rd Edition, John Wiley & Sons, Inc., New York, NY,

27 K. Kirkpatrick (2013) Software-defined networking. *Communications of the ACM*, **56** (9), 16–19.

28 B. Fitzgerald, N. Forsgren, K. Stol, J. Humble, and B. Doody, (2015) Infrastructure is Software Too. Available at http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2681904

29 Dell (2015) Technology Outlook White paper. Available at dell.com/dellresearch.

30 J. Mössinger (2010) Software in automotive systems. *IEEE Software*, **27** (2), 92–94.

31 Swedsoft (2010) A Strategic Research Agenda for the Swedish Software Intensive Industry.

32 J. Schneider (2015) Software-innovations as key driver for a Green, Connected and Autonomous mobility. ARTEMIS-IA/ITEA-Co-Summit.

33 D. Grier (2015) Do We Engineer Software in Software Engineering. Available at https://www.youtube.com/watch?v=PZcUCZhqpus

34 L. Baresi and C. Ghezzi (2010) The disappearing boundary between development-time and runtime. *Future of Software Engineering Research* **2010**, 17–22.

35 M. Salehie L. Pasquale, I. Omoronyia, R. Ali, and B. Nuseibeh (2012) Requirements-driven adaptive security: protecting variable assets at runtime. 20th IEEE Requirements Engineering Conference (RE), 2012
36 B. Omer (2014) Quantum Computing Language. Available at http://www.itp.tuwien.ac.at/~oemer/qcl.html
37 P. Selinger (2015) The Quipper Language. Available at http://www.mathstat.dal.ca/~selinger/quipper/
38 N. Wirth (1995) A plea for lean software. *Computer* **28** (2), 64–68.