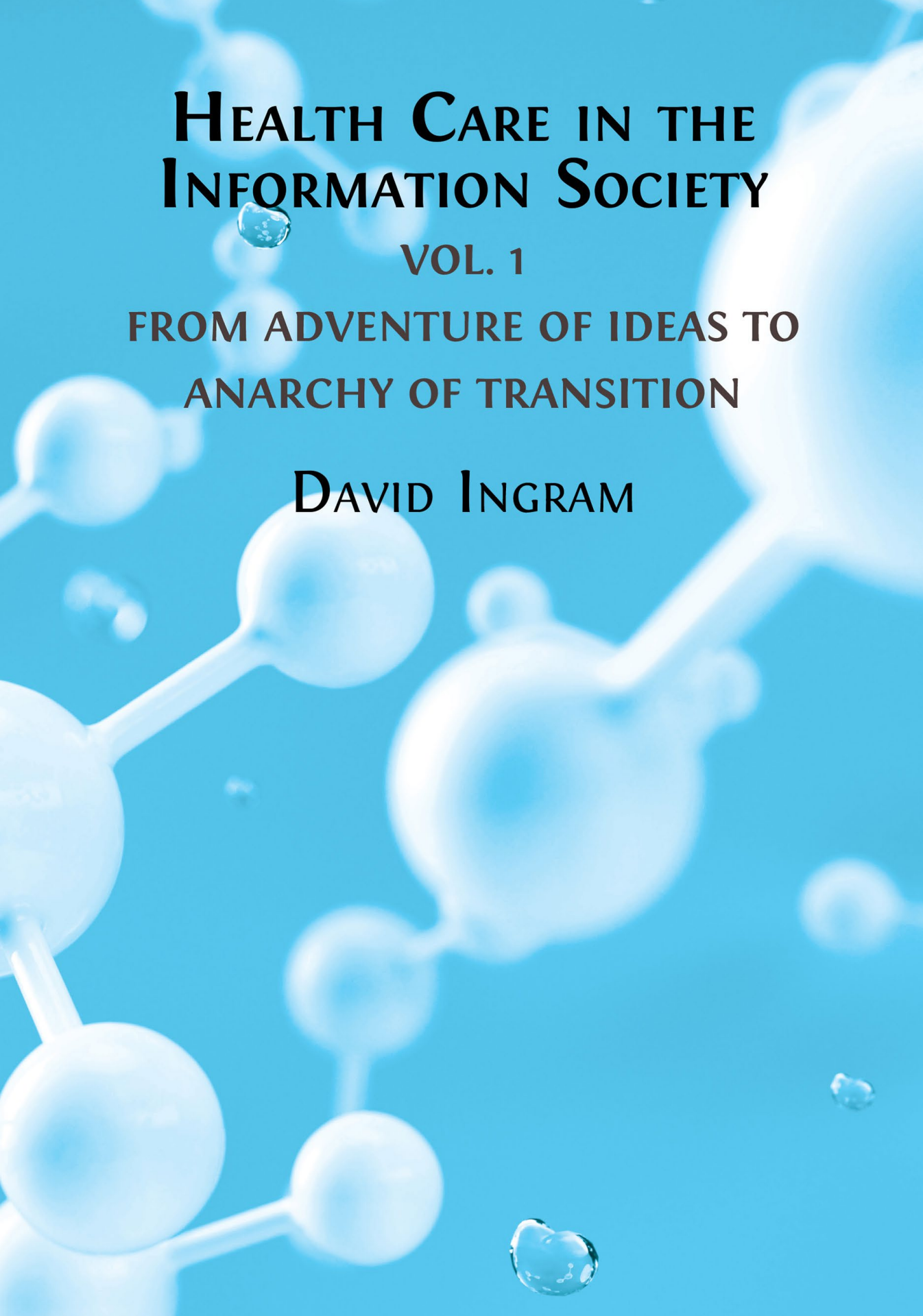
Health Care in the Information Society

Vol. 1

From Adventure of Ideas to Anarchy of Transition

David Ingram



https://www.openbookpublishers.com

ISBN Paperback: 978-1-80064-952-1
ISBN Hardback: 978-1-80064-953-8
ISBN Digital (PDF): 978-1-80064-954-5
ISBN Digital ebook (EPUB): 978-1-80064-955-2
ISBN XML: 978-1-80064-957-6
ISBN HTML: 978-1-80064-958-3

DOI: 10.11647/OBP.0335

Cover image: D koi, *White H2O Molecules* (27 June 2022), https://unsplash.com/photos/5nI9N2wNcBU
Cover design: Jeevanjot Kaur Nagpal

5. Information and Engineering–The Interface of Science and Society

Engineering is positioned at the interface of science and society. In health care, it connects the creators, commissioners and users of information systems, shaping and navigating pathways leading to success or failure in supporting the quality and improvement of services. This chapter celebrates engineers, with stories of their focus, skill and dogged persistence. I draw first on Samuel Smiles (1812–1904) and his 1881 book, *Men of Invention and Industry*, a wonderful account of engineering innovation through the English Industrial Revolution, to draw parallels with innovation in the information revolution of our age.

The chapter associates the kinds and groupings of data that are captured, processed, stored and retrieved with the devices and systems employed to do this. It describes how these have evolved, from the remote village life of my childhood, through school and university days, to my desktop today, in my now global village life, and the Cloud of computational resource that it immediately connects me with. It highlights how characteristics and limitations of devices and evolving computer programming paradigms have channelled both theoretical and practical developments, and determined their usefulness. It connects the discussion of models and simulations in the preceding chapter with data models, information models and knowledge models of today.

The chapter tracks the parallel evolution of software and algorithm, from early empirical methods closely aligned to the underlying machinery of the computer, to programming languages based on theory of data and algorithm, tuned to different domains of application, seeking tractable solutions for the computational challenges they pose. It concludes with a discussion of the standardization of computer systems and methods and the transformational infrastructure of the Internet and World Wide Web. The closing reflection, which concludes Part One of the book and sets the scene for Part Two and Part Three, looks towards a new interface of science and society, as the anarchic transition through the Information Age leads into a reinvention of health care supported by care information systems construed and sustained as a public utility.

 https://doi.org/10.11647/OBP.0335.05

> He alone invents to any good purpose who satisfies the world that the means he may have devised had been found competent to the end proposed.
>
> –Doctor Samuel Brown[1]

> Too often the real worker and discoverer remains unknown and an invention beautiful but useless in one age or country can be applied only in a remote generation or in a distant land. Mankind hangs together from generation to generation; easy labour is but inherited skill. Great discoveries and inventions are worked up to by the efforts of myriads ere the goal is reached.
>
> –Henry Mayers Hyndman (1842–1921)[2]

> The wonder of yesterday becomes the common or unnoticed thing of today.
>
> –Samuel Smiles (1812–1904)[3]

The stories and connections made in the chapters thus far have spanned philosophy, mathematics and science. These might be headlined as about musings, measurements and models! This chapter moves to another emanation–that of machine! The computer is a machine, and it is principally engineers that give it life and connect it with health care and society.

The engineering domain is sometimes described as lying at the interface of science and society–connecting the two. It is where theory and practice meet, pushing forward the boundaries of science and developing and improving technology whereby lives can improve, and society move forward. It is where material, method and construction meet, in making and doing things that serve and protect us–scaling from prototype to everyday device and method, system and infrastructure, and creating and nurturing new communities and environments where these products are used and supported. I started as a mathematician and physicist, immersed myself in connecting the computer with medical science and health care, and ended up as a chartered engineer and honorary physician, so I declare my interest in promoting this cause.

Information is sometimes described as data enhanced with added meaning and context. Some descriptions work backwards from knowledge. As previously mentioned, David Deutsch described knowledge as information with causative power, and Charles West Churchman (1913–2004) and

1 Quoted in S. Smiles, *Men of Invention and Industry* (London: Read Books, 2013), p. 50.
2 Quoted in ibid., p. 50.
3 Ibid., p .58.

Richard Ackoff (1919–2009) described information as knowledge for the purpose of taking effective action. These descriptive connections indicate cross-reference. As discussed in Chapter Three, information has come to occupy a middle ground between knowledge and data, distilled and ordered for the purpose of guiding action that determines what is then made and done. Chapter Six explores how the concept of information entered scientific discourse in the context of the search for scientific understanding of the unique nature of living systems. This chapter is about the engineering of information systems for everyday use.

In Praise of Engineers

In my book, and in this book, engineers emerge as heroes, often unsung. Elena Rodriguez-Falcon has proposed calling them 'ingeniators'–a term stressing the role of ingenuity and imagination rather than mere technical proficiency. I like the idea–Spanish, German, French and Norwegian languages follow it, and it would be good for English, too, although the word itself feels a bit cumbersome. Engineering is about making and doing. It is an imaginative approach to life and a state of mind. I have observed and followed people trained in science who have made fundamental advances in their fields. Many such people also possess the heart and skill of engineers. It is a two-way street–many trained in engineering have paved the way for scientific advances.

The father of my career-sponsoring professor of medicine, John Dickinson (1927–2015), was an engineer and John inherited energetic engineering genes. As well as being an internationally renowned doctor and teacher, in his limited spare time he was often busy working with motorbikes, cars, musical organs, central heating systems, electric typewriters and computers! Experimental physiology captivated him early in his career; he wrote his first book about the electronic circuits he had devised, empirically, for capturing physiological signals. Clinical research involving experimental treatments and numbers did not have the same practical resonance for him and he did not engage substantially in that field. His everyday clinical practice was a synthesis of art and science, education and professionalism. In his more reflective time, amidst his six-day working week, he spent much energy in keeping abreast of research on essential hypertension, on which he was a world authority, and, with me alongside, in experimenting with computer simulations of human physiology, to the perplexity of many of his colleagues!

John had prodigious energy–playing squash competitively and organ music assiduously, and attending scientific, professional and musical

events voraciously. The ennobled former Regius Professor colleague of his, sitting beside me at his Festschrift memorial meeting, asked if John had ever been put forward for a national honour. I thought likely not–in truth he would have been a worthy candidate but probably never gave such matters a second thought. People captivated by making and doing things seldom think like that.

Thus far in the book, I have shuffled on and off academic hats of mathematics and science and will now put on the hat of engineering, conferred on me, as a Chartered Engineer, by the United Kingdom (UK) Engineering Council. I did my doctoral work in the early 1970s in a mixture of academic departments of engineering, physics, medicine and computer science. My experience of engineering started between school and university, when I spent three months travelling the country as a prospective future management trainee in the heavy engineering industry. For a month, one summer, I worked in the machine-shop of an apprentice training school of a huge factory in the North-East, and out on the factory shopfloor, subject to shopfloor discipline. I learned to operate metal lathes and milling machines. One weekend, some of the many machine workers I got to know there were running a summer fair social event, where whole families turned up and showed off and shared their hobbies. They invited me along. These engineers, almost universally, had model-making hobbies–amazing model steam engines, aeroplanes and the like. A wonderful spectacle of working models and they and their families loved them. Engineering was in their blood.

Academic science envisages and explores the way forward in discipline, on stepping-stones and sometimes in timely or lucky long jumps. It is fuelled and refreshed by joy of discovery and refinement of knowledge. Engineers are artisans–building bridges, experiencing the practical realities of the world where their work is used and appreciated, sustained by joy in making and doing things that work and are useful.

There is sometimes a clash between the kinds of people who are motivated by and wish to learn by making and doing things within societal context, and those who prefer to learn at a safer distance from the societal shopfloor. Such protected places may indeed be ivory towers, but such are not confined to universities. It is good fortune, but perhaps increasingly rare, to have a working environment where the risks and opportunities of learning and practice can coexist and support one another. Engineers gravitate to places where interesting challenges present in working contexts. They tend to be robust and down-to-earth people. Maybe that is why there are few engineering pioneers visible in the higher echelons and corridors of politics and power! Engineering is hard and often unsung and unrewarded work. It exists within wide social contexts, facing multi-faceted challenges.

Engineering in Context

Problems reflecting poor understanding and a lack of practical grip have perennially bedeviled policy, design and implementation of complex engineering projects at a national level. An inherent difficulty faced is the often very wide range of contexts in which engineers must design and make things that work. First, an amusing story, that caricatures the national scene.

Sometime around 2000, I was asked to chair the national launch event for a new policy document on education and training in information technology for National Health Service (NHS) staff. A key theme was to be the joining together of disparate professional groups within a shared common initiative. The meeting was addressed by a health minister and was organized by an NHS manager who was also looking after an initiative to create a national Health Informatics Academic Forum, which I was helping to pull together at that time. A glossy brochure was prepared for the occasion and on its cover was a diagram of three cogwheels, arranged and meshed–a visual metaphor of gearbox and traction. Similar diagrams appear all over the Internet, intended as a metaphor for how different roles and functions integrate with one another in an organization or system. Here is a slide I used to use to illustrate this sort of pitch (see Figure 5.1). Some do not enmesh the cogs, thus unintentionally implying, to an engineering mind, freely spinning wheels connecting nothing with anything!

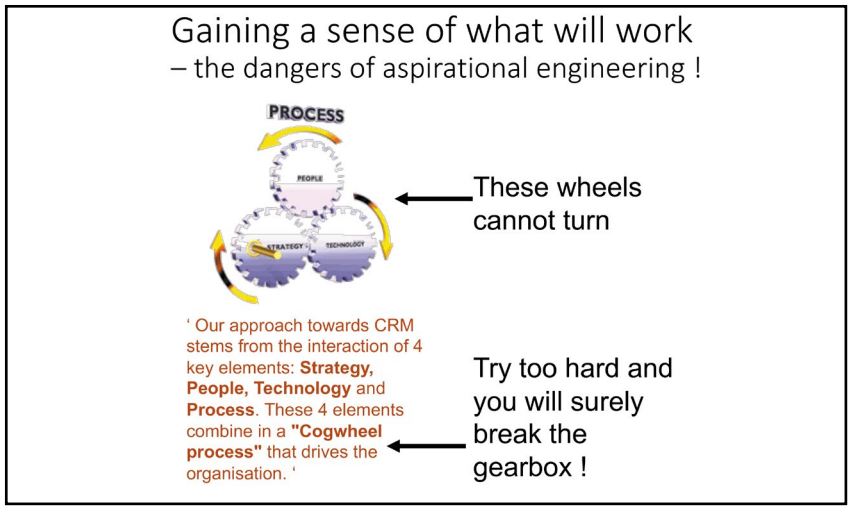


Fig. 5.1 Three gear wheels engaged like this lock together and cannot turn. This image appears frequently as a supposed metaphor of a smoothly functioning organization. It is, rather, an ironic metaphor of the widespread lack of understanding and appreciation of engineering! Image created by David Ingram (c. 2010), CC BY-NC.

At the beginning of the meeting, I glanced at the pile of brochures, one for every attendee at the occasion, and saw a problem–I mentioned it quietly to my colleague, who blanched and swore me to secrecy, which of course I respected. The problem was, as anyone who has played with Meccano, studied mechanics, made, or done anything with, any kind of gearbox would know, this arrangement cannot work. The gears are locked in a deadly embrace and torque applied to any of the cogwheels will be totally resisted by the other two. Increasing the force applied will inevitably break the machine. Rather like an early car, hurtling along (well, thirty miles per hour say, in those days!) and being accidentally thrown into reverse, causing the gearbox to explode!

Deadly embrace is a term describing a fault in program code where several threads of program logic, operating asynchronously, pause in a closed loop of dependencies, one on another. It is a set of mutually self-defeating current states and planned actions–faulty engineering–nothing happens! The metaphor of the three cogs is not the intended one for a smoothly functioning human organization. It is a cautionary metaphor for

unsuspected deadly embrace–the danger of unwittingly putting together machines or organizations that cannot, by the nature of their assumptions, specifications and design, be implemented and function as intended. Unwisely forced action in such circumstances, can incapacitate or break technology and burden or break organizations required to use the product. Most do not have eyes to look under the bonnet to observe the workings of computer systems. Many information gearboxes, lashed together and grinding, have populated the Information Age. And many have failed before launch.

Nowhere are costs sunk more deeply than into and around strategic national infrastructure, of which information infrastructure is an increasingly large component. Physical infrastructure is there for all to see while information infrastructure is hidden from view. But the strictures that its malfunction creates and imposes are widely felt and experienced, as much as are the traffic jams queued up on motorways undergoing repair. There is wasted time, cost, frustrated effort and disappointment.

There is also an underlying problem of professional status. Engineering has long been short-changed in national life. The snobberies and vanities of profession and discipline have often labelled it pejoratively as for 'nerds'; I have listened to eminent doctors describe engineers (and implicitly me!) as such. James Watt (1736–1819), who pioneered steam power–an innovation and infrastructure that changed human society for the better and for good–was one such nerd. I will tell his story, and those of similar engineers, below, as they have the power of metaphor to illuminate the story of information engineering of the past seventy-five years. Charles Babbage (1791–1871)–who pioneered the engine of information, the computer–was more erudite, but he, too, was typecast as a nerd. Given what these two faced and what they made and did, perhaps it was something of a compliment! Nerdish, as in 'not easily repressed or daunted'!

Engineering reputation typically rests on a lifetime of sustained practical achievement. Academia bestows and defends status through purity and advance of bounded discipline. Engineering is applied science, obligated to both god of discipline and mammon of practice, and thus not pure enough for the pinnacles of academic honour. The guilds of academia and society perennially debate and adjudicate the relevance and impact of academic work. Professions have feet in both camps, but trades are not considered professions. There is no Nobel prize for engineering; but none for mathematics either, so none the less admirable for that. But the engineering contributions to the scientific advances that have won many Nobel prizes, are noteworthy.

Engineering is about learning by doing. Learning requires memory–personal, occupational and institutional. Information is easily lost and

forgotten, especially, now, in its burgeoning, mutually disjoint and contradictory electronic forms. Archaeologists uncover physical evidence that guides current understanding of former times. The wonderfully preserved Roman aqueduct in Segovia is remembered and visited by tourists, like us, of today. The Information Age is not leaving discoverable Egyptian temple or Rosetta stone decorations and traces of culture and language. Failed or outdated software and digital records are easily lost beyond recovery.

My local council transferred its historic records of the city's domestic properties to microfilm. I needed to check ours and had a look–they were already faded, some no longer legible with the reader provided at the office. There is supposedly a national database–it is substantially incomplete. I suggested it might be a good idea to let the local community know about this reality, in case anyone wished to check and secure what they still could of the records of their own properties, for themselves. The council shrugged its shoulders, pleading budget cuts.

What about our medical records? Maybe we should more urgently work to enable personal possession of these, once again, as is still the practice with personal paper and film records, in many countries. As further discussed in Chapter Seven, information infrastructure, like most public utilities, is judged by how successfully it recedes from sight and is only noticed when it goes wrong. In health care, we must take special care to ensure that this does not mean when the cost of remedy is a ransom, or it is simply too late, as already with much of our property's council microfilm record.

Very few early computers and their running software are conserved in science museums. Very many products of human endeavour, building and operating the information infrastructures of today's world, have already disappeared into the mists of time. It is hard to build and sustain for posterity the engineering reputation of a Christopher Wren (1632–1673), Watt, George Stephenson (1781–1848), Isambard Kingdom Brunel (1806–59) or Gustave Eiffel (1832–1923), without leaving behind beautiful churches, steam engines, ships, bridges and towers, for later citizens to see and enjoy. This is not wholly true–Tim Berners-Lee, Larry Ellison, Bill Gates, Steve Jobs, Jeff Bezos, Elon Musk and Mark Zuckerberg will surely be long remembered by historians. There are many others who created the science and engineering underpinning their achievements.

Buried deep in all information infrastructure are many ground-changing intellectual and practical achievements, of stature comparable to those that led into the Cloud and the World Wide Web. The Turing machine and lambda calculus of computation, the interplay of engineering with mathematics and science in the development and manufacture of computer devices, the methods of programming languages and formal logic, the

relational calculus of databases, have provided foundations of discipline and infrastructure of immense significance.

Polymaths breach the defended boundaries of discipline and are not always liked for it, on either side of the battle lines. Babbage was variously mathematician, philosopher, inventor and mechanical engineer. In his later career he rose to become Lucasian Professor of Mathematics at the University of Cambridge but was first a lecturer in astronomy at the Royal Institution in London. In those times, laborious use of tables of astronomical data was required for the purposes of navigation. This spurred his inventive mind to devise machines to automate the work of creating them. The Babbage machine was dismissed with disdain by George Airy (1801–81), the Astronomer Royal of the times, in his advice to the government, when it enquired of him about its significance. He was quoted as follows: '[…] I replied, entering fully into the matter, and gave my considered opinion that it was worthless'![4] He was clearly given to airy judgements!

Stepping forward to the twentieth century, when mathematics, after such as Gottlob Frege (1848–1925), Alfred North Whitehead (1861–1947) and Bertrand Russell (1872–1970), was moving into a new era, theory of computation evolved from strong mathematical roots and credentials. And early electronic computers–valves, resistors, inductors, capacitors–arrived from a mixture of government, electrical engineering industry and academic partnerships. A new academic discipline, as computer science then was, tends initially to be classified as a sub-discipline of an existing and accepted one. Some, such as materials science, which draws together physics, chemistry, metallurgy and ceramics, become disciplines and schools of study. Computer science was originally owned by mathematics and then by electrical engineering. With its increasing academic credentials and popularity with students, it now lays claim to its own hybrid sub-disciplines, such as computational physics, computational biology and computational medicine–even bioinformatics in some places! I felt impelled to keep track of them all through the anarchic transitional years of the Information Age. Their significance and academic traction were hard to predict, and different constituencies and interests batted names to and fro. Information and informatics, as disciplines, have ramified ever more widely over all disciplines and thus faced similar challenge of identity. Connecting widely across disciplines, they have a distinctive home with none. 'What is reality', 'What is information?' and 'What is life?' have turned out to be closely connected and deeply enigmatic questions, as Chapter Six explores.

4 Quoted in G. B. Airy and W. Airy, ed., *Autobiography of Sir George Biddell Airy* (Cambridge, UK: Cambridge University Press, 1896), p. 152.

There are few if any polymaths, today, who can journey widely across the increasing numbers of connected disciplines of academic discourse, all around the circle of knowledge that has exploded in the Information Age. The lone investigator has been supplanted by the diversely and widely connected team. As now also in health care, where multidisciplinary and multiprofessional teams oversee complex treatments and care pathways. Neil Gershenfeld proposed the regrouping of academic discipline around grand challenges facing society, in which all disciplines had a part to play–ageing society, artificial intelligence, clean energy. Many Universities have tussled with the difficulty of reframing their missions in this way, while, at the same time, remaining focused on narrowly framed research metrics that emphasize identity and profile of individual disciplines. This dual approach was articulated by Derek Roberts (1932–2021), when he was University College London (UCL) Provost, himself coming from a stellar career in the engineering industry. UCL's research mission has been noteworthy in bringing disciplines together in this way. Framing and painting a picture of the grand challenges facing society, requires an institutional framework and a palette of colours drawn from across academia and across society.

The famous Maurits Escher (1898–1972) lithograph entitled *Drawing Hands* (1948) is an optical illusion that illustrates, for me, the paradox arising when depicting a grand challenge from multiple perspectives of discipline and practice–each hand is clasping a pen and drawing the other.[5] The image is a visual metaphor for the writing of individual stories about grand challenges. All disciplines and professions write the story of medicine and health care. This lithograph also emotes complementarity of perspectives in storytelling. From theory comes practice; from practice comes context and test of theory, as well as recognition of the need for and shape of new theory. Theory moves into practice and practice moves into theory. Translational medicine has often been thought of and presented as a one-way street from science into practice. Likewise, development of software has traditionally been thought of in terms of a succession of one-way 'waterfalls' from requirements, downstream to systems analysis, down again to coding, and then to a (guaranteed!) successful implementation. Software engineers learned that this does not work well, and moved to

5 M. C. Escher, 'Drawing Hands', *National Gallery of Art*, https://www.nga.gov/collection/art-object-page.54237.html. Escher used visual paradox to illustrate complex ideas. I often used this image when describing how health and care, theory and practice, health care and informatics, are co-evolving through the Information Age. It illustrates the important idea of complementarity that I highlighted in the Introduction, drawing on the immediate post-war Reith Lectures of Robert Oppenheimer. Escher's many woodcuts and lithographs, which I refer to several times in the book, are readily viewed online.

rapid prototyping and agile design, embodying the ebb and flow of design, development and implementation in practice.

If engineering fails and a bridge wobbles, buckles or collapses, the problem can be diagnosed and rectified. Living systems adapt to errors, mistakes and misfortunes–the errors of transcription and mutation of DNA, and other accidents and chance events. In their structure and through their function, they have evolved able to defend themselves. Medicine provides additional armoury, outside as well as inside the body, and care for the individual helps them towards recovery, rehabilitation and renewed self-reliance. Information is subject to malignancy and degeneration, and work is needed to keep it relevant and in good shape. The peer reviewers of science and the editors of Wikipedia perform roles of maintenance and repair. And the engineers of information systems maintain and sustain them, and keep them relevant, sound and safe.

The past fifty years have often exposed lack of capability and capacity to achieve ambitions for innovation in information systems. It has been costly learning within multiple and chaotic contexts of change. Issues of discipline, profession, organization, scale and standard became entangled and muddled. Success and failure are not well summarized and acted on in binary terms: 'distributed practice is dreadful, we will impose central control'; 'central control has not worked, we will leave it to local practice'. A bit like management of the national economy, where, devoid of more sensitive and specific control levers, policy and practice interact and the economy tends to bump along in limit cycles of oscillation.

Many problems at the policy interface of science and society pose intractable challenges; they have been characterized as wicked problems and I discuss these in Chapter Seven. Leadership in coping with and adapting to wicked problems is a uniquely human challenge. As the saying goes, 'leaders go first'. Leaders protect followers and build trust. But pioneers–who sometimes, but by no means always, succeed into positions of wider leadership–can be awkward souls. They are not always good at, or interested in, being judged by or judges of their peers. As one illustrious pioneer, Galileo Galilei (1564–1642), who suffered greatly from peer judgement, remarked: 'I would rather discover a single fact, even a small one, than debate the great issues at length without discovering anything at all'![6]

6 Quoted in D. L. Goodstein and J. R. Goodstein, ed., *Feynman's Lost Lecture: The Motion of Planets around the Sun* (New York: W. W. Norton and Company, 1996), p. 17.

Engineers as Innovators–From Steam Engines to Information Engines

I had a close connection with the contemporary worlds of heavy engineering along five years of my songline, when at university and then working in the industry, in the 1960s. Beside me today, as I write, is my grandfather's 1889 school prize; the leather-bound, now somewhat tattered edition of Samuel Smiles's (1812–1904) *Men of Invention and Industry*, cited above. Published in 1884, it is an enthralling account of people, inventions and struggles in the preceding centuries, and the contexts and communities in which their ideas crystallized, and their projects developed. It describes major changes, such as in steam power, railways, shipping, cloth making and printing, that heralded massive change in society.

These events saw determined agonists pitted against equally determined, more powerful, antagonists–defenders of *status quo* and vested interest. Their battles stretched over many decades. The final chapter of the book is a series of accounts of people Smiles had met in his travels around the country, who worked in their own homes and pursued hobbies that had risen to the level of national acclaim, one such the principal engineer of the pioneering era of fabrication of reflector telescopes, which he mastered to pursue his hobby of astronomy. The chapter is entitled 'The Pursuit of Knowledge under Difficulties'. Pioneering invention and innovation are difficult!

I have used Smiles's accounts in tracking back through several hundreds of years, to collect engineering parables: of shipbuilding and its connection with commercial, military and government establishment from the Middle Ages, and of steam power and its connection with unfolding physics, industry and transport, from the eighteenth century. As with the story of library classifications in Chapter Two, used there to give a historical context to contemporary struggles in formalizing computable knowledge, these stories illustrate features in common with contemporary struggles of pioneers of information engineering in medicine and health care.

What we often cannot recognize or discuss well, in the here and now, because too complex, uncertain and contentious, can sometimes be better framed and said more acceptably, but still authentically, in the context of parallels drawn with historical events and different domains, separated at a safe psychological distance. In making these connections, I in no way intend to compare humans with steam engines and ship propellers, although they do, on occasion, share a tendency to get a bit too hot, blow off steam and create a lot of froth! Engineering history is relevant because it is at the engineering interface of information and health, and how this is thought

about and managed, that things have often gone wrong in information for health care. There has been a lot of bubbling, hissing and scolding!

Of course, things do now play out quite differently, as well. Lifestyle today has become entrained to rapid pendulum swings of technological change, occurring in Internet time. Human minds and institutions adapt more slowly, entrained to a human dynamic that is more akin to the slowly shifting orbit of the Foucault pendulum of long ago. Today, we know that pendulums constructed with multiple degrees of mechanical freedom can exhibit chaotic patterns of motion. Ideas and their agonists and antagonists, sponsors and detractors are buffeted by events, chance or otherwise. They emerge, progress, survive and die, often chaotically.

Smiles's book features some great quotations as chapter headings. As with all such citations, I have not accepted Microsoft Word's kind offers to correct for grammar or style! Here is a bold claim from Sir Humphry Davy (1778–1829) that sets the scene–a plug for the importance of engineering, I think:

> The beginning of civilization is the discovery of some useful arts by which men acquire property comforts or luxuries. The necessity or desire of preserving them leads to laws and social institutions. In reality, the origin as well as the progress and improvement of civil society is founded on mechanical and chemical inventions.[7]

In one of his autobiographical records, Isaac Newton (1643–1727) wrote: 'It is certainly apparent that the inhabitants of this world are of short date seeing that all arts, as letters, ships, printing, the needle etc, were discovered within the memory of history'.[8] One hundred and fifty years later, Smiles remarked that:

> Most of the inventions which are so greatly influencing, as well as advancing, the civilization of the world at the present time, have been discovered within the last 100 or 150 years. We do not say that man has become so much *wiser* during that period; for, though he has grown in knowledge, the most fruitful of all things were said by 'the heirs of all the ages' thousands of years ago.[9]

Here are reflected the contemporary musings and angst of the Information Age–new inventions are changing the world extremely quickly but what matters to humankind remains as expressed thousands of years ago.

---

7 H. Davy, 'Progress of the Arts and Sciences', *The Saturday Magazine*, 416 (1838), 246–47 (p. 246).

8 Quoted in Smiles, *Men of Invention*, p. 2.

9 Ibid., pp. 1–2.

Smiles notes 'recent triumphs with electric power and electric light', but places James Watt's invention of the condensing steam engine in the front rank:[10] an invention that provided power for pumping water from mines, propelling ships and railway engines, transforming transport and manufacturing, and powering printing presses to communicate the information of William Caxton's (*c.* 1422–91) printed word.

The story of the development of the steam engine is a parable of engineering at its interface with the society of the times. In counterpoint was its interface with physics–the stories of Robert Boyle (1627–91), Nicolas Carnot (1796–1832) and Rudolf Clausius (1822–88), unravelling the gas laws in terms of pressures, temperatures and volumes of gases, and the theory of thermodynamics, linking concepts of heat, work, energy and entropy. In later times, with these properties modelled and quantified in terms of velocity distributions of the atoms and molecules comprising the gases, physics moved on to a statistical theory of thermodynamics, seeding a new concept of information linked with the enumeration of states of order and disorder in physical systems, pioneered by Ludwig Boltzmann (1844–1906). Later, John von Neumann (1903–57) progressed these concepts into the language of quantum theory and Claude Shannon (1916–2001) built on them in his theory of communication of signals, which was termed 'information theory'.

The story of steam power is rich in historical interest and insight, with parallels to the present-day story of information and health care. It was a powerful vector of transforming change of organizations and society at large, challenging entrenched thinking and assertions of status that were not ready to give way. It forced open a way to the Industrial Revolution and powered its plant. It created the railways and challenged the moguls of shipping, and their carefully guarded wealth and influence of the times. The contemporary study of organizational change and its implications for health care policy have been an interesting focus for anthropologists, such as Donald Berwick. Their observations and links with design science have interested and guided policy makers, as they sought to chart their way through the Whitehead anarchy of transition into the Information Society.

Shipbuilding, at the heart of trade and battle over empire, had been in continuous transition over centuries, from wood to iron construction, from oars and sails to steam and paddles and propellers. Opposition to innovation from threatened commercial, political and professional vested interest was a recurring theme of those times. Smiles records that Humphry Davy and Walter Scott (1771–1832), luminary figures of the age, ridiculed

10 Ibid., p. 2.

the idea of using gas for lighting, as advocated by Watt's entrepreneurial genius colleague, William Murdock (1754–1839).[11] This seems, he says, to have been the root of the term 'gaslighting', for shaming and ridiculing opponents! Murdock was ultimately awarded a gold medal of the Royal Society in 1808, for his work!

I have selected some other stories from different areas of engineering, to illustrate the feel of the times. Take, for instance, Phineas Pett (1570–1647), who proposed a radically new design of ships for the Navy in the early seventeenth century. This had shipwrights in a flurry. The Venetian galleys they were used to had evolved and were suited to calm Mediterranean seas but ill adapted to stormy seas further north. Judicial review was commissioned by the Royal Court–government and politics were royal matters in those days. Eventually, his new-fangled ship, christened The Princess Royal, triumphed, and reset ship design thereafter, as 'the parent of the class of shipping which continues in practice even to the present moment'.[12]

The first model steamboat was, Smiles suggests, made by Denis Papin (1647–1713), a Huguenot physician and Professor of Mathematics at Marburg. In 1707, he fitted a steam engine to a small boat. A more practical design was patented by Jonathan Hull (1699–1758) of Campden, in Gloucestershire, in 1736. He tested it on the Avon River nearby at Evesham. James Watt's double acting condensing steam engine of 1769 was the first power source capable of 'impelling a vessel'. It was not until 1815 that the first such boat appeared on the Thames.

The story of this condensing steam engine is particularly instructive: with Watt as inventor, Matthew Boulton (1728–1809) as promoter and sponsor (the firm of Boulton and Watt) and Murdock as developer and improver.[13] It rested on the tripod of their cooperating skills and abilities, that allowed it to make progress in design, broaden its scope, create new market, survive adversity and be sustained. Watt's engine arose from him playing with a model built by Thomas Newcomen (1664–1729). Boulton was already a successful businessman. They collaborated in business and their original application for the engine was, as mentioned above, to pump water from coal pits. Uncanny that computers likewise 'pump' data!

Murdock got involved as a young man, in 1779, sorting out practical difficulties with the pumps in use by the Boulton and Watt company in Cornwall. Smiles describes his improvements thus: 'these had William Murdock's genius stamped upon them by reason of their common-sense

11 Ibid., p. 141
12 Ibid., p. 38.
13 Ibid., p. 123.

arrangements which showed that he was one of those original thinkers who had the courage to break away from the trammels of traditional methods and take shortcuts to accomplish his objects by direct and simple means'.[14]

Watt was, by all accounts, a determined struggler–something of a tortured genius. As Smiles records:

> Watt lived on until 1819; the last part of his life was the happiest. During the time that he was in the throes of his invention, he was very miserable, weighed down with dyspepsia and sick headaches. But after his patent had expired, he was able to retire with a moderate fortune, and began to enjoy life. Before he had cursed his inventions; now he could bless them. He was able to survey them and find out what was right and what was wrong. He brought his head in his hands to his private workshop and found many means of enjoying both pleasantly.[15]

Smiles prefaces his sixth chapter on the inventor of the steam-printing engine, Frederick Koenig (1774–1833), with a quotation from Daniel Defoe (*c.* 1660–1731): 'The honest projector is he who having by fair and plain principles of sense, honesty and ingenuity, brought any contrivance to a suitable perfection, makes out what he pretends to, picks nobody's pockets, puts his project in execution, and contents himself with the real produce as the profit of his invention'.[16] If perhaps idealistic to today's ear, this captures Smiles's admiration for the character, commitment and staying power of inventors such as Watt and Koenig.

The struggles over steam power continued, on multiple fronts. The story of George Stephenson (1781–1848) and the railways, and the powering of factory machines, is well known. I highlight, here, the contentious battles over sea power and naval conflict on land. An innovation of that era was John Harrison's (1693–1776) chronometer, for use in determining longitude at sea[17] (the story of which I told in Chapter Three, in context of the impact of innovation in measurement). As with Babbage's engine, it was the eminent Astronomer Royal of the time who covertly opposed and obstructed the idea, in this case being interested only in lunar tables. It took forty-five years for the invention to gain parliamentary approval, in 1773.

14 Ibid., p. 145.
15 Ibid., p. 146.
16 Ibid., p. 156. I have known someone remarkably like that, who died too young, a couple of years ago. He was my career long colleague and friend, Jo Milan (1942–2018), whose endeavours in creating the information systems at the Royal Marsden Hospital, in England, enabled and accompanied its progress as a centre of excellence in cancer care. I tell his story in Chapter Eight.
17 Ibid., p. 104.

His wooden clocks ended up in the Science Museum in South Kensington and four chronometers are housed at the Royal Observatory in Greenwich.

Other controversies raged over ships. Smiles describes a Dr Lardner (I take this to be Professor Dionysius Lardner (1793–1859)), who had argued in a Royal Institution lecture in 1838 that coal- and steam-powered ships, that were struggling to be accepted, would never cross the Atlantic because they could not carry enough coal to raise the steam required.[18] The feasible marriage of electrically powered vehicles and weighty battery technology power sources are debated in similar terms, today! He also writes that Sir William Symonds (1782–1856), 'the surveyor and principal designer of Her Majesty's ships, was opposed to all new projects. He hated steam power and was utterly opposed to iron ships. He speaks of them in his journal as "monstrous". So long as he remained in office, everything was done in a perfunctory way'.[19]

Steam power triggered similar letting-off of steam in a quite different kind of innovation–the steam printing press. This attracted venomous opposition from printers of the times, as did computer typesetting in our era. Smiles records the secrecy surrounding its initial introduction: 'Great was the secrecy with which the operations were conducted. The pressmen of the *Times* office obtained some inkling of what was going on and they vowed vengeance to the foreign inventor who threatened their craft with destruction'.[20]

The idea of the propeller had first been conceived by Watt around 1770. He described it in a letter to a friend as 'a spiral oar'; it became known, derisively, as the 'screw'. This term had, from the 1600s, been associated with application of pressure or coercion, echoing the rack of torture. Few merchant ships were built and fitted with the screw up until 1840. The Admiralty was strongly opposed and made slow progress in adapting it for the Royal Navy. There was, however a very determined developer of the propeller screw, Francis Pettit Smith (1808–74), who faced down the Admiralty's obstruction antics, albeit at considerable personal cost. He succeeded in the staging of an experiment. Paddle steamers and screw-propelled boats were put into a racing competition and the propeller triumphed. As Smiles writes, 'Francis Pettit Smith, like Gulliver, dragged the whole British fleet after him'.[21] Eventually, the Admiralty had to give in.

Commenting on the struggle, Smiles observes that Smith derived no dividend for his invention: 'Smith spent his money, his labour and

18 Ibid., p. 3.
19 Ibid., p. 70.
20 Ibid., p. 169.
21 Ibid., p. 70.

his ingenuity in conferring a great public benefit without receiving any adequate reward and the company, instead of distributing dividends, lost about £50,000 in introducing this great invention, after which, in 1856, the Patent Right expired'. He comments on the determination that had been required, as follows:

> Sir Francis Pettit Smith was not a great inventor. He had, like many others, invented a screw propeller but, while those others had given up the idea of prosecuting it to its completion, Smith stuck to his invention with determined tenacity and never let it go until he had secured for it a complete triumph. As Mr Stephenson observed at the Engineers meeting, Mr Smith had worked from a platform which might have been raised by others, as Watt had done, and as other great men had done; but he had made a stride in advance which was almost tantamount to a new invention. It was impossible to overrate the advantages which this and other countries had derived from his untiring and devoted patience in prosecuting the invention to a successful issue.[22]

The political establishment later caught up. Robert Stephenson (1803–59), a Member of Parliament, convened a meeting in later years to commemorate Smith's achievements. Also illustrating the support that comes only after such contentious and disruptive battles of ideas have been won, Smiles quotes Baron Charles Dupin (1784–1873), who compared the farmer Smith with the barber Richard Arkwright (1732–92), inventor of the spinning frame: '"He had the same perseverance and the same indomitable courage. These two moral qualities enabled him to triumph over every obstacle". This was the merit of Screw Smith–that he was determined to realize what his predecessors had dreamt of achieving; and he eventually accomplished his great purpose'.[23]

These stories have common features. They are parables for our times about the challenge of engineers and engineering and the struggle to innovate. They are stories of invention and traction, talk and distraction; of inventors who innovate and create, and detractors who block and procrastinate; of those who join in and those who observe and judge. If there are three priorities that should be learned from this history, by any who would aspire to create and innovate in health informatics, they are implementation, implementation, implementation–in making and doing things; creating and sustaining necessary new communities and environments; scaling and standardizing methods. And in all this, learning by doing. This was the maxim I adopted for openEHR, which features as an example of a

22 Ibid., pp. 71–72.
23 Ibid., p. 72.

mission for clinically grounded, technically rigorous and professionally and organizationally engaged engineering in health informatics–the subject of my parenthetical Chapter Eight and a Half.

The Information Technology Industry

This section is a rapid scan along the timeline of the evolving information technology (IT) industry–snapshots more than full video. It glosses over some technical details that are introduced, more systematically, in later sections. Its aim is to illustrate the depth and scale of evolutionary advance in the nascent IT industry, where many technology teething problems have spilled over into, and interacted with, wider disruption and advance of health care.

Automation of clerical work using mechanical and electronic devices, called Hollerith tabulator machines, was the innovation that gave birth to the first International Business Machines (IBM) Corporation. It was founded by Thomas J. Watson Sr. (1874–1956), one hundred years ago in 1920. Caxton's printing press metamorphosed into an electro-mechanical information automaton. IBM Worldwide Trading Corporation was established in 1949 and taken forward from 1952 to 1971 under the leadership of Watson's son of the same name, Thomas J. Watson Jr. (1914–93). The business based on Hollerith machines, pioneered by the father over the preceding half century, metamorphosed, under the leadership of the son, into mainframe computers and commercial data processing software for automation of clerical work. This strategy, that had been doubted as viable by the autocratic father, triumphed under the diplomatic son!

This was the era of change and transition after the end of the Second World War in 1945, when a new international order emerged under the auspices of the League of Nations. In America, East and West, and in England, two embryonic worlds of computation evolved and enmeshed–those of the hardware and software of computing machines. Two wider worlds also enmeshed and engaged in this endeavour–the commercial world of computers and tabulators and the world of academic science and engineering. In America, this history played out around the evolution of the IBM computer.

The new hardware evolved rapidly over the next decade, starting with thermionic valve-based designs that were the successors of the ENIAC machine of the early 1950s, and moving on to transistor-based machines from the mid-1950s, based on William Shockley (1910–89) and colleagues' Nobel Prize-winning advances in electronic technology. The IBM series of mainframe computers spread into the commercial world and the military,

underpinning the space missions of the 1960s. A key design focus of the early machines was in finding the best trade-off between what capabilities should be built into the machine as hardware, with an early focus on hardware to perform floating point arithmetic, and what should be the preserve of stored programs running on the hardware. The design of software to run on these stored program machines played out in the evolution of two pioneering languages, FORTRAN and LISP. The former name arose from FORmula TRANslation and the latter from LISt Processing. FORTRAN led the way in the world of numerical computation and LISP in the world of symbolic reasoning.

IBM created and positioned itself at the epicentre of major technological change. It amassed huge wealth, starting with large mainframe computers and, over many decades, adapted and metamorphosed its product line into aggregations of smaller and smaller, minicomputer and microcomputer machines, and larger and larger software platforms and research and consultancy services. It has been the great survivor. Microsoft, Google, Apple and Amazon have likewise lifted on these currents of transition into the present-day Internet Cloud resources of the Information Age. The race now, in university and industry laboratories, is for quantum computers, where the dream is of tens of quantum qubits, now stuttering into life, emerging on a larger scale, with computing power matching the billions of nanoscale transistor circuits lined up in the largest of today's devices.

I recall being told the story of a customer who purchased an IBM mainframe in the 1960s and later chose to upgrade it to the next machine up in the range, which computed much faster. The new contract was signed and had a suitably hefty price tag, doubling the power of the machine. A support engineer arrived to perform the upgrade. He switched off the power, removed a cover on the side of the sizeable machine, reached inside and, with a pair of pliers, cut a small metal link between two components on one of the circuit boards. He put the cover back on, switched on the power, ran some tests, and departed! It may be completely apocryphal–I have no means of knowing–but it says something with a ring of truth about the disconnect between cost to provide and price to purchase, in the world of IT!

IBM pioneered new technology in many domains, including health, where the Watson software, commemorating IBM's founder, is a machine intelligence guru of medical knowledge today. The Digital Equipment Corporation (DEC) pioneered the smaller scale minicomputers of the 1960s and 1970s. It grew into an international conglomerate and made similar efforts to innovate in health, in niche areas of database (such as the MUMPS language), imaging and laboratory systems, by giving customers the wherewithal to create bespoke systems interfaced with the devices they

made and used. Its co-founder, Kenneth Olsen (1926–2011), was another mogul of the times and became wealthy Massachusetts aristocracy.

A story told to me by one of DEC's salesmen of the times, went as follows. One year, Olsen addressed the company's public meeting of shareholders, shortly after a major power failure had occurred, plunging the East coast of the USA, including the DEC home base near Boston, into protracted darkness. This led to a noticeable population boom, nine months later. Taking questions at the end of his talk, a small lady sitting near the front raised her hand and asked: 'Mr Olsen, the electric grid breakdown has been such a disaster, can you reassure me that it wasn't our computers that caused it?' To which, it is said, Olsen replied: 'no mother, it wasn't!'

Many companies–Control Data Corporation, Sperry-UNIVAC and Honeywell in America, ICT in the UK, Bull in France, Siemens Nixdorf in Germany and more–competed with IBM at mainframe level. Control Data put major effort into a large-scale computer-assisted learning system that they christened Plato, with a graphical interface that was both innovative and appealing to users. The venture quickly folded once the company's priming investment had been expended. A mainframe of the scale of the CDC7600, dedicated solely to educational courses, was beyond almost anyone's scope for investment. Just as well, as such courseware was highly experimental, limited to CDC systems, and soon obsolete.

Many more companies–Hewlett-Packard, Data General, Modular One, Norsk Data and more–competed with DEC at minicomputer level. The minicomputer manufacturers were selling into an engineering-literate customer community, and these people wished and needed to know their products inside out. They deployed the increasing power of the minicomputer to provide general purpose multiuser systems for small communities of users, such as the HP 3000 set up introduced at McMaster and the early PDP-11/45 system that I later introduced, configured and operated for a while, at St Bartholomew's Hospital (Bart's).

I was developing software through these early years: largescale simulation programmes and clinical applications in radiotherapy, nuclear medicine, and neonatal intensive care records, starting on the IBM mainframes of UCL in the late 1960s, then on the DEC minicomputers of the University College Hospitals (UCH) in the early 1970s, and at Bart's in the 1980s. The early work in medical physics involved the writing of machine code software and the construction of hardware required to interface hospital physics devices with the minicomputer, as well as the adaptation of the computer operating system software to accommodate the high data transfer rate of some of the imaging systems. Developers needed to know and work on all these components of the computer system. This breadth of knowledge and

skill had to be acquired alongside the work of developing and operating the clinical applications the systems were used for.

The wider market developed differently, leaving the DEC approach of offering highly customizable computer systems, focused on technically knowledgeable customers and users, increasingly outmoded. Device manufacturers had gradually caught up and incorporated digital signal processing and data management circuit boards within their own products. The mid-1970s were a key turning point in the market. The industry started to offer machines and software tools and packages that were accessible, not just by specialist programmers but by anyone with the wish and perseverance to build their own applications. These users required no knowledge or insight into how the machine and operating system were designed and functioned–just the ability to follow rules in using them. Like the progression from early cars, where every function and need had to be carefully monitored and catered to, to cars that people just fill up with fuel and oil, inflate tyres, drive, park and occasionally clean!

The pace of advance in computational power and data storage capacity of semiconductor devices was characterized in Moore's Law, reflecting new technology of miniaturization of chip manufacture–from companies such as Intel, Texas Instruments, Zilog, Acorn and AMD.[24] The Unix operating system was born in the early 1970s at the pioneering industrial research Bell Laboratories.[25] Unix became established as an operating system for minicomputers used in science and engineering, implemented across different manufacturers' machines.

Database design evolved more coherently and generically after Edgar Codd (1923–2003) succeeded in formalizing rigorous set-based algebra and practical heuristics for representing and managing complex data relations. Prior to this, it was principally the physical properties and limitations of magnetic tapes and spinning discs that shaped approaches to data storage and retrieval, leaving little scope for optimizing these methods according to the structures of the data themselves, and how they would be used in

24 Gordon Moore was a co-founder of the Intel chip manufacturing company. In 1965, he asserted that the number of transistors that could be fabricated onto a silicon chip would double every year over the coming decade, and the cost of computers would fall by a half. In 1975, he revised the estimate to doubling every two years. With some fluctuations, this relationship has held through to 2020. It looks set to continue for some years more, with nanometre scale size of transistor being demonstrated this year.

25 The original Bell Company was named after Alexander Graham Bell (1847–1922), the Scottish founding father of practical telephony. Ownership now rests in the Nokia Group. Bell Labs has been a stellar performer in technological innovation, with nine Nobel Prizes arising from its historic work.

practice. Diverse database management methods came and went. Oracle reigned supreme in this software revolution and business–the wealth of its founder, Larry Ellison, later devoted to his yacht-racing passion. Network technology advanced in parallel over several decades, from Arpanet to Ethernet and Internet, from browser technology to World Wide Web to client-server architecture, from parallel and GRID to Cloud computing. As oncoming waves, these advances created new turning points in the market.

The major computational infrastructures for physics, astronomy and biology advanced, with government investment and drawing on a scientific community culture of cooperation and sharing of key components (although there were, of course, rivalries!). The teams innovated to improve network, processor and database architectures, and software methods, where commercial products of the time fell short of meeting need. Innovation embodied in the robotic arm that is central to operation and maintenance of the Culham prototype fusion reactor, near Oxford, emerged within that kind of science and engineering community. The design of reactor and robotic technology were parallel and interrelated challenges, proceeding together–a shared journey of discovery.

With software, the focus evolved from packages to mega-suite applications, supplied expensively by mega corporations. These became one-stop shops for business applications–Systems Applications and Products in Data Processing (SAP) for the organizational back office. Oracle started to sell mega-suites for managing large institution back-office applications, such as finance systems. The cost of adapting such products to suit the needs of the client institution was typically high. Variations in user needs were ironed out through procurements that included consultancy and training, to persuade and support the purchasing organization in changing its working practices, rather than leave the onus resting on the software supplier to adjust its product to meet those needs–sometimes beneficially and acceptably, but sometimes not. Changes in information systems and related working practices could take months and years to bed in, running alongside existing systems and then supplanting them.

Faults in software are continuously identified and fixed, and new ones created. Continuing support contracts for system hardware and software, to keep them functioning and updated, became a necessity for the user and a reliable source of revenue for the software supplier. The supplier's business model shifted from outright purchase to leasing.

Likewise in device markets, with customers tied into a proprietary brand of product, the cost of operating the device (for example, buying a supply of idiosyncratically designed ink cartridges for a printer) became more expensive over the device lifetime than the original purchase cost of

the printer itself. These evolutionary trends were mirrored in the world of health care IT systems, as I explore further in Chapter Seven.

In the hospital, the institution was stirring and responding to the computer suppliers knocking on their doors, with futuristic sales pitches designed to entice and secure orders. The building of local team capacity had started in the physics departments, which were active in supporting clinical services where clear new roles were foreseen for computers–for example, in clinical measurement, radiotherapy, nuclear medicine and clinical laboratory services. Computerized patient administration systems were a major focus of interest of the times. Information technology became a bastion that enhanced and reinforced the power base of management professionals in their perennial battles with clinicians at the coalface, in the running of health services. They had allies in government and the politics of increasingly expensive and unaffordable health care service delivery and IT became ensnared in power struggles over central and local policy and directive, and clinical professional autonomy. This was where the big money for IT innovation in health care came to reside and where mega-projects arose.

Health care, and public sector organizations, more generally, switched their attention towards knowing how best to buy IT from suppliers and manage large contracts with them. This was, itself, a complex and impactful interplay, but it lessened the focus on knowing how to innovate, and catalyze and lead change, in the growing range of IT systems needed for effective and efficient delivery of services. Scientific and technological advance, fragmentation of incompatible IT systems, and fragmentation of health care services went hand in glove through this era. And few focused on where IT fitted in the development, upskilling and support of the evolving and future interdisciplinary and multiprofessional health care workforces, in adjusting their working practices to this new world, thus increasing their already heavy burden. I recently had my Covid booster and seasonal flu vaccinations. The jabs took a couple of minutes at most, including answering a few simple questions about my health, and me adjusting my sleeve. The completion of screens of computer questionnaire, entering information about me, already many times known, to many systems, in disparate health care contexts, by the friendly clinician, took about ten minutes!

The outsourcing of services has provided easy targets for blame when failure looms! But outsourcing can be a very risky strategy when it involves procuring a service that you, the purchaser, do not understand and cannot articulate in a satisfactory way. ‘Any colour of car so long as it is black’, Henry Ford’s supposed motto, indicated power of the producer to dictate; ‘no one got sacked for buying IBM’ indicated the protective attitude of the wary purchaser of computers. They reflected newly enforced realities and

defences of the day. Focus on the wholeness of health and care services suffered as a result.

By the early 1980s, desktop microcomputers with graphical user interface became the norm, taking over from text-based display devices. New competing empires of microcomputer and operating system arose. The machines started as small boxes, with keyboard, cassette or floppy disc drive for program and data storage and attached television screen–such as the Acorn BBC micro, Sinclair Spectrum, Atari and Commodore PET computers. Prices came down to hundreds of pounds. They were both cheap and accessible, stimulating a domestic consumer market for systems running games and simple applications for word-processing, accounting and database management. They also gave opportunity for novice users to experiment with their own simple code–inspirational for school children and hobbyists of the era. Today, the Raspberry Pi has taken the cost of such apparatus down to tens, from hundreds, of pounds.

IBM returned to see off its established minicomputer and fledgeling microcomputer challengers, plunging into the microcomputer era of the 1980s with the IBM PC. DEC tried but could not quite make this transition. I bought one of their early microcomputers, the DEC Professional, for my work on educational software and interactive videodisc systems. I spent many weeks implementing the Mac Series software on top of its cumbersome operating system. I quickly switched to the IBM PC and the MS-DOS operating system. Apple and IBM took their time and learned their way to dominance of the desktop computer market. Dedicated word processors became the 'must have' equipment of every office, disrupting the previous pattern of secretarial services.

Microcomputer operating systems progressed from the simple functionality of CPM and DOS to Microsoft DOS (MS-DOS) which evolved to several stages of Windows, and of Apple iOS, consolidating experience in early minicomputer operating systems. Microsoft operating systems were licensed to many manufacturers and suppliers, to use for running their computer systems, packaged with office software and rudimentary databases. Apple kept hardware and software inhouse within an integrated offering–in time creating a proprietary smartphone technology stage and charging actors for the right to perform on the platform it provided. Linux, an open-source Unix-like operating system and its Ubuntu derivative, colonized desktop, server and autonomous device domains. Android and Ubuntu arose as open-source operating systems–Android to colonize smartphone devices, pitching as rival to Apple's proprietary iOS.

Applications software development became the expensive part of the business and the hardware and its operating software a buried utility. Programming languages and tools migrated across different manufacturers'

systems. Intellectual Property Rights in software were difficult to defend, street markets became the office front of software piracy. Eastern Europe, although constrained and held back for decades by limited resources and trade embargo, applied brain power and persistence to the cloning of the computers themselves. There was a weirdly amusing episode that I encountered about one such cloning.

Around 2000, I was invited to Timisoara, in Romania, to give a keynote lecture at a national meeting. On the following evening, I found myself in conversation with the local University's head of computing services. He told me of the pride his team had experienced during the isolation of the country in the Ceausescu era, in succeeding in cloning the DEC PDP-11/34 computer to produce a more powerful home-grown version. He told me an interesting story concerning the uprising, which ultimately displaced President Nicolae Ceausescu, catalyzed by the brave public stand of a Lutheran priest in Timisoara. There had been demonstrations in the streets of Bucharest, close to the central telephone exchange building. The exchange was controlled by the PDP-11/34 surrogate computer, which was the pride and joy of the regime, symbolizing national triumph over its enemies.

Concerned that the crowds would invade and destroy the treasured machine, the regime placed a tank outside the exchange, to defend it. Inside the exchange, the computer was ticking away, acting as a social media telephony hub of the uprising, connecting, and communicating events and coordinating tactics across the country. The regime was oblivious–their actions defeated their own attempts to contain and isolate the protesters! The complex world of IT is fertile domain in which to let loose unintended consequences!

I have had similar experience of the practical engineering skills and persistence of physicists in Poland, in building and operating a radio telescope, during the country's locked down years. It still runs, lovingly maintained by staff who have dedicated their careers to preserving its machinery and maintaining the electronic and machining materials and skills needed in support of their science. It is a treasured artefact of its times, housed, alongside other classic optical telescopes of earlier times, at a hillside observatory near Krakow; our friend, a physicist who runs the radio-telescope, took us to explore it.

Returning to software development, the needs of research, and concern for independence from commercial constraints of software copyright, reinforced cooperation on software for data analysis. Collaborative developments of software packages for statistical analysis of research datasets stemmed from the 1960s–designing algorithms to migrate mathematical methods that were formerly enacted with pen and paper and aided by pocket calculators, into ubiquitous software. Some such endeavours spun into businesses

with their products marketed internationally. New statistical methods that required substantial amounts of calculation, which had not previously been feasible to perform, became tractable with the use of software packages. What had previously engaged 'Huxley summers' of devoted wheel turning of hand-operated calculators, to solve their Nobel Prize-winning equations governing the nerve action potential, surrendered to a few milliseconds of computer muscle power.

There was a downside, of course! Any user, not just a trained statistician, could set the machine to churn through huge numbers of calculations, quickly and precisely. 'P fishing', or 'data dredging'–combing datasets to uncover correlations deemed statistically significant–became something of a plague. The boundaries of statistical significance and practical significance became quite blurry in some disciplines. The five-sigma threshold level of significance used in physics in hypothesis testing, to render a result worthy of designation as a discovery of new knowledge, would rule out significance of the results from all experiments in biology and medicine! Such would be completely inappropriate, of course, but much that passes as significant research finding in clinical studies–when reflecting on money spent and benefit realized–seems hardly worth knowing. And as science spreads far beyond laboratory experiment into population studies, the experimental biases implicit in design, conduct, analysis, interpretation and dissemination of the work have rightly come under greater scrutiny–selection bias, detection bias, observer bias, publication bias and more. All these assumed greater importance as the scale and range of data science started to explode.

Many advances in information technology have had their origins in academia and moved swiftly to richer and better funded environments in private industry. Harvard University had brief early connection with the founders of Microsoft (Bill Gates and Paul Allen (1953–2018)) and then Facebook (Mark Zuckerberg); Stanford University was the starting point of Google (Larry Page and Sergey Brin). The UK e-science program was an exciting era of cooperation in which I collaborated and helped to oversee. This and many similar government-funded programmes have shared learning with industry-funded and -led research laboratories, underpinning fundamental technological innovation, and gaining wide commercial traction across the world–the Cloud arose in that way. In recent years, Google has absorbed and hugely progressed ideas of artificial intelligence that were nucleated in academic environments, and developed ground-breaking quantum computing technology. These kinds of synergy have mirrored long-established pioneering establishments in the USA, such as the Bell and IBM labs.

Today the provenance of Cloud-based technology has focused applications development on web technologies. Many overlaying layers of 'technology stack' specialize and standardize subdomains of design, such as the user interface. A new legacy has started to raise its head–that of the diversely evolving patterns of programming languages and their specialization to different kinds of computational task, and the generation, testing and maintenance of code for these. Faults (bugs) that creep from the system and program design stages into operational systems are hard to detect, diagnose, and correct. As they interact with other computational processes, they can accrete incremental noisy complexity and vulnerability.

Good system design and poor program implementation is clearly problematic. Poor design and competent program implementation–in other words, done by the books–can be equally troublesome. I have seen an operationally fault-prone design of a software system for posting patients' laboratory test results to their electronic records, albeit correctly coded, proving vulnerable to an unforeseen contingent 'event' and sending some results to the wrong record. In this case the event was an accumulation of too long a queue of results held up in the system before they could be posted to the associated record. I have not heard that a programming error was the cause of the faulty control of the early Boeing 737 MAX aircraft. It was, one gathers, due to oversight of an unforeseen contingent event, which led to a design vulnerability. The aircraft relied on a single sensor to activate software to correct for incipient instability encountered during flight, to which the plane was more vulnerable because of an implicit design imbalance in its flight trim; that, in turn, arising from technically suboptimal positioning of its engines. These concerns had, apparently, been overridden in its subsequent manufacture, due to commercial pressure to maintain the aircraft's production schedule and contain its costs. From this picture, one might imagine a potential vulnerability in the operation of an autonomous health care software system, in the context of the contingent variability which is so characteristic of individual health care 'events'. How can we be sensitive to, mitigate and guard against such risk? We should remember that that is what human clinicians seek to do every day. That is why they must be interested in the patient's story as well as their data. The record must centre on the individual patient and capture and communicate its clinical meaning.

An urgent problem today is how to cope with the accumulating and costly legacy of obsolete code and methods of coding. The solution must involve writing and communicating better code. Mutual coherence of system design with programming language and method is fundamental. In the world of software technology, this evolving quest has devolved into two competing factions–functional programming and object orientation.

And new buzzwords abound for platforms that themselves write code for applications, with a hundred or more of what are called 'no-code' or 'low-code' environments.

It will not surprise that I am concerned about the entropy of legacy systems code created along this anarchic pathway. We cannot escape the fact that the battling of entropy that naturally accumulates in this way involves continuous work, and sometimes disruptively so! And it will also not surprise that I have found the domain of health care to be almost paradigmatically prone and vulnerable to this entropic disease, and thus a place where we must place special emphasis on the mutual coherence of our efforts to computerize. Not always popular with legacy landlords of decaying properties, but true.

Data Processing

A key message of the Information Age is that quality of health care depends increasingly on how well its information systems connect and deal with both the syntax and semantics of data–how they are structured and what they mean. This resonates with the words of Florence Nightingale (1820–1910) quoted in the Preface, from a hundred and fifty years ago. This may sound superficially obvious, but its underlying practical implications have taken many decades to be understood and sink in. The meandering course of the marriage of health care with information technology over the past five decades has been manifested, expensively and consequentially, in the failure to understand, capture and manage data well–coherently, consistently, conveniently, sustainably and in context. It is not a surprising history–this has been an anarchically changing, complex and contentious period in which to plan, navigate and learn the nature and significance of data, in the context of understanding and promoting health and combating disease, and providing the care services that are needed.

Learning from the experience of those decades is important if information systems are to avoid emulating the crisis of the world's monetary system in 2008. Information pandemic is prospectively overwhelming. Much of what we connect with *in silico*, and depend on to persist, will wither in time. As clay tablets and papyrus weather and wither, so too do digital media. Information technology is no more immortal than life itself. Semiconductor junctions decay, and the silicon crystals used to make them will, no doubt, eventually merge slowly back into the sand they came from. Our virtual worlds are mortal, too!

The next section recalls the history of electronic data storage technologies and introduces data models that link theory and practice in the design and

implementation of database systems. Data are often described as a sea. We slip from classical correctness in treating the word as a plural noun (*data* from *datum*) into a modern usage of data instead of datum as a (singular) sea. I will no doubt continue to slip between the two–my meticulously insistent classicist, Latin and Greek scholar school headteacher has long disappeared from my shoulder! The following section adopts a parallel approach to knowledge models and knowledge-based systems. The story then moves to theory and practice of software, as algorithm and programming language.

The engineering challenges encountered in using hardware and software to represent and integrate data, information and knowledge, in systems that embody observation and measurement, database and knowledge base, record keeping, logic and computation, extend beyond these elements into considerations of information models and information architecture. These in turn lead to consideration of software systems and software standardization, within and between different domains of application, with the goal of achieving coherence and meaningful computational interoperability within and between information systems. The final sections address system architecture and the drive for standardization, which have been greatly influenced by the advent of the unifying framework of the Internet and World Wide Web, and the global information infrastructures and user communities these have enabled to gain traction, develop and grow.

Data Storage

Data models and information models feature in the design and formal specification of databases used for storing and accessing data. They operate at different levels of abstraction. The former is focused on the logical arrangement of data, to facilitate its rigorous and secure storage in the machine database. The latter on the characteristics and flow of data, in the context of services and organizations that the database supports. The database system is software that relates logical model to physical layout of data within the storage medium–managing secure and efficient access and maintaining integrity of the data stored.

As discussed in the previous chapter, models serve purposes, and modes and methods of modelling reflect those purposes. There are not intrinsically right or wrong ways to model data. It can be done accurately or inaccurately, reliably or unreliably, well or badly, using different methods that are proposed and find use. Alighting on successful ways to model data requires a combination of insight, experiment, and experience. The past fifty years have been highly experimental.

I do not seek, here, to catalogue the many paradigms and methods of data modelling and storage that have featured along my songline. It would serve no great purpose and the text would quickly disappear into the thicket of anarchy, obsolescence and confusion that has characterized the domain. It would be a headache to write, as much as to read. What seems more relevant is to focus on the story of how the different paradigms and methods arose, the purposes they served, the compromises they represented and the diversity they displayed. I will trace a path through this evolving story.

Information systems are programmed in software and there is therefore an intimate connection between the physical representation of data within systems and the algorithms, computer languages and programming paradigms employed. Information systems also support reasoning about data and knowledge, making inferences and guiding decisions in the context of both data and knowledge. There is thus also intimate interaction between representations of data and representations of knowledge, and the software that programs and enacts the reasoning connections made.

All this is a long story and following and absorbing it requires mental persistence. Information engineering requires another kind of persistence–the term has been appropriated there to mean storage of data on a physical medium. Device technology, data persistence method and database performance–data in and data out–are closely coupled, but deeply below the surface of what a user sees. Unfortunately, but rather inevitably, it is often a matter of 'out of sight, out of mind', and the consequences of that short-sightedness can be profound. The unfolding story starts with the engineering that underpins data storage devices.

I have pondered from time to time over many weeks, now six months into writing this book, over how to tell the evolving story of data storage, data models and databases, along my songline. It has cropped up in many places in the first drafts of the ten chapters and not very satisfyingly. As a fifty-year scan quickly reminds one, the topic has been a muddle; one that students of today, seeking skill in the domain, would be well-advised not to attend to in any detail! The standard approaches to description and organization of data are coursework and textbook stuff for the student of today but they were an anarchy of discovery and change in the making. Technical detail of methods employed has been captured in classic textbooks, such as the well-known one by Christopher J. Date–his 1975 textbook now into its eighth edition–but these quickly date.[26] There are increasingly clear and structured overviews of the field in Wikipedia. The story has continuing impact on and

26 C. J. Date, *An Introduction to Database Systems* (Delhi: Pearson Education India, 1975).

interaction with health care information systems, and these are the aspects I focus on here.

I am sitting down to reorganize the material for this section on a Monday morning, and it is the first morning I have overslept beyond an early start. I came to, knowing that the matter had been drifting around in my dreaming consciousness through the night. I awoke, decided upon introducing it as a story of my childhood village community, village data and everyday devices, seventy-five years ago at the beginning of the Information Age. Zobaczymy [we will see]![27] The kinds of data stored today have much in common with those of this childhood, although utterly different in variety, scale, methods employed and purposes served. What has happened to these data in the intervening years? How were they and are they stored and accessed, now, and where are they?

In my childhood in the small village in rural Hampshire, maybe five hundred people lived there. There were similar, even smaller hamlets dotted around in the countryside, with the nearest small town five miles away. There was a village shop, which doubled as the bakery and post office, a primary school for fifty pupils, coming from the surrounding area, a church and church hall, a grocer, doubling as a much-frequented sweetshop, a pub, a farmyard and a woodyard, and the village bobby's (police) house. Stately homes and estates of country squires and landed gentry dotted the surrounding landscape. Prince Philip and pals were sometimes at play with guns and dogs, in the fields surrounding the twenty acres occupied by the children's home run by my parents. A game keeper appeared with a brace of pheasants after each shoot, as a reward for us children not straying to disturb the birds and shooting.

The village data was written down in numerous forms. Simple financial accounts featured in running a household and sat alongside diaries, address books, letters, lists and documents, with notebooks acting as catch-all *aide-memoires*. A small amount may already have been recorded onto early magnetic tapes. At the start of my songline, financial data recording the income and expenditure of a business was recorded in books, often huge ledgers. Data was entered in tabular form, recording details of individual transactions as they arose, and later summarized in sets of accounts, drawn together and presented in a standard manner so they could be viewed and independently verified by auditors, as full and correct representations. The accounts and records covered employment of staff, purchase of material items or services, income from sales of goods or services. From these were derived further tables recording assets held, profit and loss, taxation and

27 On this Polish expression, see Preface.

the like. These would be just one part of the documentary record required by law. Documents and correspondence by letter were filed in hefty racks and cabinets.

Imagine me, now, as a social historian, arriving to learn about and document village life, as in Gilbert White's idealized *Natural History of Selborne* (1789).[28] Such a history is often remembered and told through stories. The aim is to collect as much data as possible to complement and support those stories. Who lived there? In what families? When were they born? Where did they live? How did they earn their living and how much did they earn? Why did they choose that way of life? These are: who, what, when, where, how and why questions. They are used in navigating the provenance of lots of different kinds of data and their contexts.

Waves of data flowed, pervading and proliferating in the village. In the school: school classes, teachers, classmates, attendance records, exam results, the annual school play in the village hall. The village shops, pub, bobby, church and church hall, even the farm, and wood yard, all have stories to tell and records to keep, connected one with another through people and contexts of events that they share–the villagers, extreme winter snow or summer drought, harvest festival, outbreaks of contagious disease. They form narrative accounts and each person, and every family, has stories to tell. Over time, these play out further and connect more widely, within and beyond the village.

Much of this history is an oral history, as told, sung and persisted along the Aboriginal songlines. The data persist: as written and printed words, as symbols and numbers, as media, and in association with features of the landscape. They are evidence in court: presented by professionals, told, listened to, read, made sense of and adjudicated, through the eyes, ears, thought and experience of judges and jurors. They are assimilated nearer to home: in family and kinship, culture, practice and life of the village. *Family and Kinship in East London* in the 1950s, was one of my dad's inukbooks, from his difficult and impoverished childhood there.[29] The human authors of the recorded data, and their authenticity and authority, wax and wane and become established and trusted, or forgotten, over time.

Times changed and technologies for recording and communicating data shaped and were shaped by those changes. Data was communicated in greater amounts and more quickly, to and from afar, through broadcasts, letters, newsprint, books and travel. Clerking of entries in ledgers had earlier given way, in larger businesses, to recording on punched cards that could

28 G. White, *The Natural History of Selborne* (London: Gibbings, 1890).

29 M. Young and P. Willmott, *Family and Kinship in East London* (London: Routledge, 2013).

be printed from and sorted. Mechanical typewriters arrived in the village in similar very small numbers as did early cars. In my childhood, there were just two or three car owners in the village, many bicycles, quite a few horses and, of course, many Shanks's ponies (of travellers on foot!). Few houses had phones and television was equally slow in arriving. My great aunts living in a tiny Cotswold Hills hamlet, had no electricity and used oil lamps and solid fuel. Some hand-operated calculators assisted the tallying of numbers for financial accounts. Dictaphones accompanied doctors on their rounds, keeping an *aide-memoire* on the move, for later transcription to paper by a secretary. Files of correspondence, reports and accounts proliferated, and administration became data heavy and ever more complicated. Postal services ferried and telegraphed data from person to person, organization to organization, and place to place.

And Pegasus took wing as the name of an early vacuum-tube computer, programmable to process text and calculate with numbers. Computers received program instructions and data, punched onto paper tape and card, and read into the machine by tape and card readers, calculated and printed out in new forms. Program and data were also read and written, using flexowriter, which doubled as typewriter and slow printer, operating noisily to print ten characters per second. Postal telegram gave way to telex. The computer had a small memory store that could be addressed and worked with directly, providing space for both program and data. Outside the machine, the data was stored on card, paper tape and as printout. A laborious workflow was tended to by teams of computer operators, processing the work in discrete batches.

There was another industry evolving rapidly and in parallel–that of radio and television, recording and broadcasting for the entertainment industry, on sound and then video media. Records of music had long been based on mechanical devices–folding paper card for pianola and rotating spiky disc for musical chimes. This merged into the world of electrical transducers of mechanical vibration, vinyl record, stylus, amplifying horn; and then loudspeakers arrived. We had one of the early phonographs at home, beloved by my parents; they had much of Mozart's music on inflexible early records, played at seventy-eight revolutions per minute (rpm), that cracked rather too easily. In my mind's eye, I can still see the horn. And with the advent of electrical recording onto magnetic tape media, tape recorders became a rapidly growing new domestic market.

The recording and editing of such tapes, reel to reel, became a home-based hobby. Dictated oral history and record found its way onto magnetic tape. Editing–adding to and erasing bits of the record–was a time-consuming labour of love! Data was recorded sequentially along the tape, and the tape scanned with playback machines that could skip fast forwards

and backwards to locate content. The tape player incorporated a counter that ticked up the number of rotations of the tape cassette as it recorded onto or played from the device. Positions of pieces of music on the tape were written down in a notebook index. Of course, as the long tape wrapped around the spindle, the length of tape signified by each tick upwards on the counter, increased in proportion to the increasing diameter of spooled tape. This index did not signify amount of music, just its start position, and that with decreasing precision.

On vinyl records, the music was recorded and sensed as a time varying mechanical indentation along one long track–helical along the surface of a cylinder or spiral from outer rim towards the centre of a disc. The physical recording of a given loudness, frequency and length of note (*p*, *f*; B-flat; crotchet or minim) was associated with different physical magnitude, frequency of indentation, and distance along this track. Along a magnetic tape track, it was recorded and sensed as a change in strength of magnetization of the tape. In contrast to the disc and stylus arrangement, and in like manner to the phonograph cylinder, this was recorded uniformly, always occupying the same length of tape for the note recorded. In all these ways, characteristics of device and recording technology determined quality of performance–in this case the oral quality of the sound. Some still hold vinyl preeminent, although the entertainment industry has long been built on the shoulders of digital devices and systems, and now of Cloud-based streaming services.

Calculation and computation moved from mechanical (Babbage engine and hand-operated calculator) and analogue (Napier's bones, slide rule and analogue computer) devices into the electronic era. Magnetic tapes became the leitmotiv of digital data storage. One track of audio changed to seven and then nine parallel tracks of binary data on reels of magnetic tape, spinning on large tape-drive devices attached to computers. Programs running in the computer controlled this device and transmitted data to and fro between tape and computer memory. The data was organized on the tape in sequential files–one following the other down the tape–and the tape was annotated with electronic markers, indicating the beginning and end of files and first and last positions of the tape where they were recorded. Detection of these markers alerted the device hardware about where files were positioned. Software indexes were used to facilitate jumping forward or backward from one to the other. Indexed sequential files, and fixed and variable length records were introduced as new persistence methods, to adapt to changing needs and speed of access.

Writing and reading data involved spinning the tape to the position required and transferring to and from there, bit by bit, byte by byte and file by file, rotating the tape drive forward a step at a time under program

control. The tapes grew to hundreds of feet in length and electronic circuitry was devised that searched for the markers, thereby enabling a program command such as 'skip forward' or 'skip back', to the start or end of the next file or the start or end of recorded information on the tape. In these operations, the tape could be spun past the read/write sensor very much faster than when inching a step at a time, to write and read data.

So far, so good, but as with the manual editing of audio cassette tapes, or the splicing of paper tapes used for the input of computer programs and data, the sorting and editing of material on the tape under program control became a significant block on productivity. Allocation of space on the tape, and recovery of space freed up after data previously stored there had been either erased or edited and repositioned in revised form elsewhere, was time-consuming and laborious. If a file is deleted, is that section of tape still available for reuse? In charming technical language, this discarded data is termed 'garbage', and recovering and recycling the space it occupied, for further use, is termed garbage collection! Garbage became a term of derision in castigating misuse of data–Garbage In, Garbage Out (GIGO) became a catch phrase.

The deletion or repositioning of data and reassignment of the storage space freed up had to be recorded as an update to whatever index the program was using to identify and locate data recorded on the tape. Material that had been placed sequentially, with associated items located near to one another, became fragmented, as files were split into sections and recorded in non-adjacent sections of the tape. Reconnecting the separated pieces of a file required for use in a computation involved waiting for the tape to spin (described as a latency period), and this could slow the writing and retrieval of data considerably. Managing the positions of files on the tape, with multiple edits, deletions and re-recordings, made for laborious program software. Often the simplest approach was reel-to-reel editing, as with audio tapes, copying from one tape to a fresh new one, which could be done at greater speed, leaving out the pieces no longer required and adding new pieces as the tapes spun by.

The programming effort required to accommodate and adjust to the characteristics of the recording device again became a significant matter of tail wagging dog. The purpose of the exercise was not simply to achieve and facilitate accessibility of data and data storage–it was to compute with the data, and this involved searching through, calculating with and reordering the data, such as by time, place or person it described. Optimization of these data-processing tasks involved an interplay between properties of devices and properties of the data themselves. Ingenious mathematicians and engineers devised new algorithms to improve the efficiency of such manipulations of data; the utility of each pitched against characteristics

of the device for which it was implemented. Theory of data-processing stretched theory of software and database design.

As far as the data were concerned, magnetic tapes were essentially automated filing cabinets. The digitized files comprised text and numbers, occupying pigeon-holes on the tape, termed fields and grouped in records of fixed, sometimes variable length, and records grouped together in files. The file held groupings of data from lists and tables previously recorded in paper documents. Where the purpose was to keep all the data in the document together, it made sense to design storage and retrieval, document by document, accordingly. Where the purpose was to work with the content of multiple documents–accessing and processing pieces of data and updating other stored documents, accordingly–granularity and flexibility of access within records was a more complex challenge.

At the time of my first practical encounters with mainframes and minicomputers (in 1969, on London's first Master of Science (MSc) course in computer science), tape and disc technologies were equally poised–I worked with and programmed them both.[30] Pictures of spinning magnetic

30 Computing was a puzzling newcomer in the academic world and especially so in a medical school. To stabilize and find my bearings in my initial foray into this new world, I decided to enrol to attend a pioneering Masters course in computer science, at the then quite new London Institute of Computer Science (ICS) in Gordon Square. This was a London University Institute, not connected to any College at the time, directed by Richard Buckingham (1911–94), a particle physicist and mathematician, turned computer scientist. The combination of lectures and practical work was a congested curriculum. The luminary Peter Kirstein (1933–2020), credited as a founder of the Internet for his work in creating the ARPANET network link between the USA and the UK and early banking networks, was the very bright lecturer in systems programming. He owned a DEC PDP-15 in the basement and much coursework for him involved machine-code punched with flexowriters onto paper tape, read into the machine and used to explore the coding of operating systems and device drivers. With London University's decision to disband the ICS, the staff were reemployed to establish new departments at the six principal Colleges of London University at that time. Peter took a small core group of the staff into nearby UCL, first as part of a new joint Department of Statistics and Computer Science, and subsequently as the first Head of a newly created Department of Computer Science. In later years he collaborated with my Department at UCL, CHIME, in piloting the Internet IPv7 protocol in medical applications, as part of a research project in the EU Framework Programme. Keith Wolfenden—dry and slightly lugubrious in manner, but generously warm-hearted—was the lecturer on data processing. He took us through sorting algorithms, such as bubble-sorting, and we wrote and tested programs. The vagaries of processing data to and from magnetic tape were expounded—very much an example of device technology considerations dominating data manipulation in the balance of theory and practice. I learned, there, perhaps for the first time, how closely software engineering methods were coupled to the practical properties of devices. True of theory and practice of

tapes were iconic images of the times but magnetic tape technology soon ran out of steam in keeping pace with the scope and scale of data-processing requirements. However, they were not rendered obsolete and were still in use in the petabyte data stores established at places like the Central Computer Laboratory of the Research Councils (CCLRC) laboratory at Harwell, which I was shown when attending advisory board meetings there in the early 2000s. The robotically controlled modular array of tape cartridge drives was used to archive the increasingly massive datasets of e-science that were processed by powerful connected grids of computers. The backup would have required many days if carried out across a network connection and was instead achieved by placing tape cartridges in a white van and driving them to a secondary backup centre elsewhere, we gathered!

But just as tapes and tape recorders for analogue audio recordings had given way to more flexible, robust and manageable vinyl records and record players, magnetic drum and disc stores arrived in the computer room, storing megabytes of digital data. Such devices had been experimented with in the world of immediate post-war crystallography laboratories, where the world's first drum store was claimed by an inventor working with John Bernal (1901–71), friend of John von Neumann (1903–57), at Birkbeck College in London, as mentioned in Chapter Three. The storage of digital data, serving the needs of audio and video media and computation, spawned the compact disc, DVD, floppy disc, video disc and dynamic RAM data sticks.

---

measurement, more generally. This was quite tedious stuff, and I was delighted to find a higher-level language, APL, which coped elegantly and satisfyingly with the mathematical aspects of such algorithms. The mainframe of the day was the Ferranti Atlas, which occupied most of the ground floor of the Institute. Its design, which included extraordinary thin metallic strips of firmware code, was lovingly described in lectures on machine and operating system design, by Alan Fairbairn. Eric Nixon introduced us to computer science through the vehicle of a virtual machine he had devised which was programmed and ran on the Atlas. We were introduced to theory of computation by one of the research fellows, in a course encompassing Turing machine and the language of set theory and formal logic. Other lecturers covered circuit design, operating systems, and programming languages. We did coursework that introduced us to languages such as FORTRAN, PL/1, and ALGOL, but much of the time we wrote native machine code. These were early days, with none of the structural and conceptual clarity, expressive power and efficiency of today's programming languages and their platform implementations to support program design and testing. Languages tuned, rather pragmatically, to the needs of different domains of application were, and for coming decades continued to be, an exploratory maze. They sought to balance evolving theory of the design of programming languages with the changing requirements of the domains in which they were applied.

With the new devices arose new ambition to manage increasing scale and complexity of data, as well as a new challenge in developing better means for optimizing them to work well. New data storage and management methods led to stronger bonds between data structures and software. Data modelling became a mainstream preoccupation and concern. As the doyen of software paradigms Niklaus Wirth was to write in 1976: 'Algorithms + Data Structures = Programs'.[31]

Moving on from magnetic tapes, my experience in 1970 progressed to digital data stored, uniformly, in concentric rings, on spinning magnetic discs. As with magnetic tapes, there was access delay (latency) in rotating the disc to the required positions for reading and writing data, although considerably less than for spinning tapes. The discs were in continuous rotation and latency was determined by speed of rotation and time for movement of a sensor arm between the periphery and inner regions of the disc, much as a stylus reads from a vinyl record. Drum-like rotating storage devices had an array of read/write sensors positioned parallel with the axis of rotation, enabling faster access to larger data stores, but at a cost in terms of the extra hardware and control circuitry required.

Allocation of space on these devices, and recovery of space freed up, was a similar optimization challenge, and an index of the space allocation enabled the device control software to move the head directly to access the required data. Initially the discs were fixed in place within the device. Quite swiftly, technology improved to the point where discs could be housed in demountable cassettes, thus achieving an archival function akin to magnetic tapes on racks. In due course, disc technology was miniaturized, and its storage capacity expanded; floppy discs and minidiscs became consumer items at home.

Early disc devices were very temperamental–most prototypes are. Their electrical circuitry and physical components were annoyingly prone to failure and sometimes to spectacular and damaging crashes of the read/write sensors–destroying sensor, spinning disc and stored data. Risk of corruption of sections of disc and data meant that great care was needed to keep backups and test and compensate for the prevalent presence of blemishes in the magnetic surface. My first disc storage device was of suitcase size, incorporating a thirty centimetre diameter spinning metal disc which stored just thirty-two kilobytes of data–it seemed quite impressive at the time! It was always going wrong, however, and a maintenance engineer took several days to come to sort things out, bringing the bulky oscilloscopes and

31 N. Wirth, *Algorithms + Data Structures = Programs* (Englewood Cliffs, NJ: Prentice-Hall, 1976).

signal generators of the maintenance trade of that era. Essential recurrent maintenance contracts became an expensive budget item.

Design and performance limitations restricted most databases to storage of files similar in structure to those stored on magnetic tape. Searching within these files involved software for locating them on disc, reading them into memory, record by record, and analyzing the data, field by field. It was all down to the program one wrote. Products with more elaborate functionality began to emerge as disc technology settled and became more reliable. And early pioneers were already hard at work, exploring new horizons. Constraints of computer memory size forced economy towards concise, albeit not easily humanly readable, program code. Constraints of storage device performance motivated exploration of new methods of data persistence. Requirement to accommodate multiple simultaneous users of a computer, working in different domains of application, led to a need for new optimization among program algorithm, data storage and the operating system software of the computer itself.

Exploratory medical physics applications of the era worked at this interface, linking imaging devices, such as nuclear medicine scanners and cameras, with minicomputers to capture their fast data streams, to process and generate higher quality and dynamic images. And most significantly, the Massachusetts General Hospital Utility Multi-Programming System (MUMPS) was conceived and developed in Boston, in the late 1960s, in the laboratory and team of the doyen of medical informatics, Octo Barnett (1930–2020). It led hospital information systems into the Information Age and has endured, underpinning principal hospital information systems and many other domains of IT systems today.

As the speed and capacity of disc storage increased, and ambition rose, attention switched to characteristic organization of the data themselves, when determining data storage requirements and methods for accessing and processing them–issues of type, interrelationship and scale of data, and of their context and usage. It was an era of experiment and learning–prototyping, refining, discarding and generalizing. From it emerged new theories and models of the logical structure of data, independent of its chosen means of physical storage. Theory of algorithm advanced alongside experiments in programming. Theory of computation paralleled advances in processor and network technology.

Data Model and Database

The term data model came to mean an abstract model that sets out a description and logical organization of data elements, defining their

characteristics and how they are related to one another. It is an abstraction in the sense of existing independently of the technology used to store and access the modelled data.

Pursuing the village data analogy, the physical compilation of lists, financial accounts and other records became increasingly burdensome. First hand-written, stored in ledgers and filing cabinets, later dictated to a secretary, or captured on an audio device and typed out, it became an increasingly expensive process. Requirements for sharing, analysis and regulation of data expanded; this led to their being organized and expressed in different ways, and increased the number and size of sets of data required to be kept in the records. Keeping records up to date occupied more time.

The school and church kept written records. These would be descriptive of people living in the neighbourhood and the many kinds of issues and events connecting with their roles and activities. In a general way, providing answers to questions about who did what, when, where, how and why. Much of the content might well be common between these two sets of records, but, being designed and maintained independently, it was likely to be expressed differently and include different levels of detail.

The management of changes in interrelated sets of records brings new complexity. In my secondary school years, I was given the task of organizing and keeping a diary of all the school sports fixtures; setting dates and times; negotiating with all the different schools involved to make their different diaries fit together; organizing transport; and handling problems arising when, for whatever reason, the pattern broke down. We might nowadays call this a workflow–the work was mine and there was good reason to call the sports diary a 'fixture' list. We fought like crazy to keep the pattern fixed from season to season–competing with certain schools in certain weeks of the term, each year! As soon as one fixture broke down and could not be played on the usual date, in the usual location and at the usual time, the knock-on effects ramified through many school fixture lists.

It was hard work to adjust the pattern and make sure it was still consistent with every match required being played by the end of the season, with transport, availability of referees, refreshments and so on, all arranged. As a mathematical and procedural problem this was difficult enough. As a human problem, with the many possible reasons calling for changes–weather, school term dates, special events, illnesses, personal whims of teachers–a good deal of interpersonal brokerage and persuasion was required. It was a highly contingent world–imagine from there to health care service diaries!

Nowadays, we might call this process 'logistics'–in my experience, the process was not formally all that logical! A timetabling automaton would struggle to cut the mustard in resolving such matters, without much human

brokerage of the compromises required to reach agreement, each issue being felt differently in different schools. 'The computer says no' would come to rule the roost!

These rather simple examples reveal underlying complexity in the challenge faced in modelling data, especially where purposes served are shared purposes. Translated from paper to the computer domain, such models must be clear, coherent and consistent, both in the definition of the data they embody and in how they are understood, communicated and used. Human language is inexact and brokers differences and uncertainties. The computer domain is unforgiving–such limitations propagate as errors. To compensate, the usage of language on the human side of the communication becomes progressively narrowed and appropriated to more specific computer domain meanings. And concern for precision of communication of the meaning and context ascribed to data leads to ever more detailed recording of data about data–metadata. In his book *Homo Deus*, Yuval Noah Harari describes the debasement of human language and communication to fit the needs of consistently computable data, a data religion or 'dataism'–a term first used by David Brooks in the *New York Times* in 2013.[32]

Continuing with the computerization of village data analogy, data were seen to be of different kinds and groupings–there were numbers and character strings, and villagers, families, school classes, church attenders, shop customers–unbounded and ever-changing groupings. Times are seconds and years, weekdays and anniversaries, before lunch and during sleep. Locations are the home, the school, the shop, the bus stop. The groupings exist in hierarchies: children within family, school class within school. And such hierarchies break down and become blurred in their precision and usefulness.

Here natural language starts to depart company with and give in to computer-speak. In appropriated use of the terms, each 'element of data' within any such 'grouping' is called an 'instance' of that 'grouping' and the 'grouping' is called a 'class'–hence 'school class' as a 'class'! 'Class' and 'instance' became appropriated terms to describe such 'groupings' more generally, as did the word 'classification', to describe how its 'members' were determined. Each 'school class' is an 'instance' of the 'set' of all 'school classes' in the school. Each pupil is an 'instance' in the 'membership' of a 'school class'. My primary school Years 1 and 2 'school class', taught first by Miss Broadhurst and then by Miss Simpson, is where I was first

32 Y. N. Harari, *Homo Deus: A Brief History of Tomorrow* (London: Random House, 2016); D. Brooks, 'Opinion | The Philosophy of Data', *New York Times* (4 February 2013), https://www.nytimes.com/2013/02/05/opinion/brooks-the-philosophy-of-data.html

'instantiated' as a pupil of Woolton Hill Church of England Primary School, in the village of Woolton Hill, in Hampshire, in England... the computer is a hard task master in its insistence on normalizing language. I will give in and not attempt too great a precision in my normalization of these terms, not being (or for my purposes here, not needing to be) too pernickety about them.

In this way of structuring and reasoning, one person's set of data can feature in many data instances within the ensemble of groupings (classes) used to model the village data. This hierarchy of data and classes might be modelled as analogous to tree structures of the village data, with successive groupings ramifying (branching) one from another. The village data is represented in this model as leaves on trees. To search for the data we seek, we must climb and navigate the tree, through its hierarchy of branch and twig groupings, to arrive at the leaves of data we seek. But it is not always useful or appropriate to atomize data in fine detail. There are many ways of doing it, serving different purposes, and represented as different tree structures. And some groupings of data are best kept together within a single document and the document stored and retrieved as a whole, prior to any further consideration and computation. There can thus be many trees modelled in this village data wood. One can easily lose sight of the wood, faced with the tangle of trees.

The analogy of modelling data hierarchy as a tree may fail us on other counts as these are often not normal branching trees. Branches can combine as well as diverge, and a network of connections is another way to model and populate the links between the different groupings. The network may be a highly regular structure which we think of as a lattice or a highly variable structure where the more complex mathematics of graph theory may come into play. And the data tree hierarchy can, in the mind, be uprooted and inverted, to be climbed the other way up, starting from some element of data and discovering the groupings and locations where it is instantiated. For 'content addressable' file storage, a new kind of index is needed, like an alphabetical book index that points to pages in the book where the indexed topic or detail is described.

Experiments in data modelling and data storage progressed on multiple axes of information engineering, treating the data as: entity and attribute; document; tree, network, lattice and graph. These brought new richness of linkage and relationship among the data and associated new complexity in methods to keep them rigorous, up-to-date and secure. As new application domains were explored, device features and limitations became less dominant in determining and constraining database design; however, requirements to represent diversity of structure, content, extent and context

of the records themselves, that changed over time, became more ambitious and thus more difficult to accommodate and implement.

Any more detailed description of the many different paradigms of data structures that have been explored, and their aetiology, would quickly ramify into a book itself. Some have evolved and endured, and others have disappeared or been superseded. The hierarchical paradigm is still used for the Microsoft Windows Registry. However well they are defined, though, data models cannot hope to accommodate the full semantic richness of the real-world data domains they seek to represent, just as formal logic struggles to accommodate the verbal language nuances of human argument. But data quality matters in everyday life, irrespective of whether computerized or not.

The exploration of different paradigms of data modelling has focused new light on data quality more widely, informing a clearer sense of the balance that must be struck between rigour, expressiveness and ease of use of information systems. The inefficiencies and harm arising from poor or inadequate discipline in the capture and handling of data, and the cost penalties these impose, is today more apparent. What is less appreciated is that this pattern persists also in the domain of algorithms, which come together with data models in the representations implicit in computer programs. There is considerable and costly accumulated legacy of obsolete software. This issue is further addressed in the software section of the chapter, below.

A full, rational and consistent data structure, fit and performant for all potential purposes that the data may serve, is readily understood to be often unachievable, and would, in any case, need to adapt and change over time. In the village, the church decides what it wishes to computerize in its records–about births, marriages and deaths, about families, baptisms, confirmations and tithes–and models its data accordingly. Others, if so minded or required to, also structure and keep their own different records. Some stick with paper records!

However, what happens when a 'Who?' question is asked by the village doctor–such as 'Which of the children are in this Sunday class and that school class?' And why would the doctor need to seek this information? Perhaps to check against their medical records to see who has not had their measles vaccinations, after a child who is a member of both these groupings shows up one morning with early signs. Maybe a bit far-fetched–bush telegraph among parents at the school gate would likely have got there before the doctor. But the general issue here is that data exist within multiple groupings and contexts, any of which may prove relevant to a question that needs a consistent and trusted answer.

Over time, personal data came to be seen as the property of the data subject and an ethical obligation was recognized, which became an increasingly closely defined legal obligation, that they be kept up-to-date, consistent and confidential, used only for purposes for which that person had given informed consent. Thus, the need for rigorous and robust standardization of data became a new kind of both moral and legal imperative. And providing the means to meet these new requirements encountered further complexity when different aspects of a data subject's personal data were being held in multiple disjoint databases, conformant with different data models and handling their access control security differently.

When reflecting on these intrinsic complexities of handling data in computer systems, combined with concern for its confidentiality, it is perhaps not surprising that medical records have persisted so long in paper form, insecure though this may prove to be in reality–at risk of physical theft as much as cybertheft. The combined complexity of these multiple sets of requirements has been nowhere more vividly exemplified than in personal health care records.

And it was through pioneers in medicine of the 1960s and 1970s–people like Barnett, Neil Pappalardo and Howard Bleich (1934-2021), in Boston, at the Massachusetts General Hospital and Beth Israel Hospitals–that groundbreaking innovation occurred. The MUMPS language was conceived of, developed and commercialized, and used to build and operate early clinical department and hospital-wide computer applications and information systems. Jo Milan (1942–2018), my colleague at the Royal Marsden Hospital in the UK, pitched in alongside. The MUMPS-based systems perform extremely flexibly, efficiently and powerfully in clinical context, to this day, and MUMPS spread and persisted in other domains–notably in records of banking transactions.

From its origins, MUMPS had proceeded from a simple and pragmatic set of choices. It treated a persistent data store as a virtual array addressable directly by a MUMPS language program as a global program variable. To the programmer, it was like a coiled snake of extending magnetic tape, directly addressable anywhere along its length. The MUMPS language was implemented as a 'program interpreter', working line by line through the program code, and enabling direct and efficient control of the still very limited power and capacity of the machine environment, for multiple users. It implemented the storage of the snakelike data array on disc (its 'persistence', as the term became appropriated), such that programs could efficiently access and retrieve data from this array, even if data instances along the snake were located few and far between on the disc surface, and only very sparsely populating the virtual storage array structures that the language and program provided for. This was a major advance of the times.

By inspired and original use of indexing keys, it populated these sparsely occupied arrays such that, for example, elements 1, 10 and 1000 in the array only used three storage locations, and array elements could be inserted, edited and removed equally simply, with access to each element achieved via a quick and continuously 'balanced tree' search strategy, called a 'B-Tree', adjusted over time to keep database access operations optimally efficient.

A legacy of the fixed structures of data records implemented in early databases, utilizing indexed magnetic tape and disc files, was program styles and capabilities that were inflexible and inefficient, when the records were highly varied and changeable, in structure and extent. MUMPS gave the programmer direct control of computer, disc-based data and processing algorithm. It fell short, as became clearer in time, in not providing for different data types and data relationships. This was compensated for by merging a fully relational database modelling approach (which I discuss next), on top of a MUMPS-style programming environment and data persistence engine. This was the major achievement of Milan and his team, at the Royal Marsden Hospital, as described in Chapter Eight, when discussing the ETHOS software that they pioneered. Their achievements speak for themselves. As a persistence engine, MUMPS has not been surpassed in the world of leading hospital and financial systems. As an innovation it was a *tour de force* of achievement and impact. It passed the so-called 'ACID test' of database transactions–atomicity, consistency, isolation and durability–guaranteeing integrity of data despite program errors, power failures or other mishaps.

From the late 1950s and in parallel with the, in some ways similar, evolution of MUMPS, the CODASYL consortium (the Conference/Committee on Data Systems Languages) worked on specification and standardization of a network paradigm data model and language for manipulating data in records. This was the era of COBOL (Common Business-Oriented Language) that hit a sweet spot of ease and simplicity for much of the industry that was writing finance applications at the time, and it endures in many applications to this day. The trend though was towards separating the description of data structure from the language used to manipulate data. Edgar Codd's 'relational model' fulfilled this purpose, arriving a decade later and quickly becoming the dominant database paradigm in the industry. The term 'relation' thereby acquired specific new meaning in the language of, and reasoning about, data structures and databases.

The relational model proposed a mathematically rigorous and generic pattern for describing data, anchored in the mathematics of sets and the logic of propositions and predicates. Anyone following the relational paradigm shared this one rigorous theoretical model. To produce a database to meet their individual needs, they organized their data using the same

relational database methods. To do this well was an art, much as proving a mathematical theorem is an art, and some are better at it than others.

To return to the village school data computerization analogy, there are data about teachers on the staff, subjects taught and lessons delivered to consider. To organize data descriptive of these lessons, it is necessary to have a means to identify each of them uniquely in the database–this mechanism is called a 'database key'. For example, every lesson might be assigned a number to identify it uniquely. For every lesson, various fields of data–the teacher, subject and other details such as its time and location–are to be stored, referred to individually as 'data attributes', each of a particular 'data type'–number, text string etc.–and together as a grouping called a 'tuple'. A convenient way to think about relations was as a two-dimensional table structure, with attributes as columns and tuples as rows. The set of tuples populating this table formed the body of what was called a 'relation' and within the 'calculus of relations', each relation was a named 'variable'. Mathematically speaking, each row is a 'proposition', and the relation is a 'predicate'. Theorems arising in the mathematical manipulation of these variables are expressed in a 'relational algebra'.

The school needed to record other kinds of data about teachers, pupils, school classes and subjects. To avoid 'redundancy'–for example recording the same details of teachers and pupils in multiple contexts, such as in school, year group and school class membership lists, for example, as well as in the list of lessons delivered–such data was organized once, within separate teacher and pupil relations, again with tuples uniquely identified. Special relations are then created providing means to join and cross-reference between these separate relations, expressed as mathematical manipulations of the sets of data described in the associated table structures.

This was bread and butter mathematics but required careful attention to detail. Incautious design risked inefficiency and error in both the recording and manipulation of the associated data. Associated with Codd's relational model was a set of rules tying down the forms in which tables were to be designed and linked to model the village data, rigorously and safely. These stipulated practices to guarantee integrity of storage, retrieval and maintenance of the data instances that comprised and populated the records held within the physical medium that embodied the database.

Refinement of the relational database domain was further complicated by requirements to compute with the data held in the database. Dates and times might need to be expressed and viewed differently in different contexts. In some cases, it proved simpler and more efficient to store the data in a standardized Julian date format and provide algorithms for its translation to meet the different display formats required. This was more efficient and less prone to error than recording the same time in different

ways in different places. Database software started to incorporate stored programs for such data manipulations as this, that were commonly required. It was in keeping with new thinking of data as 'data objects', comprising both content and methods associated with manipulating and sharing that content, within and between objects.

The database software implementing and processing data structures conformant with the relational model, provided the interface between the abstract form of the tables and the physical layout of data thereby stored on and accessed from the storage medium, according to the patterns defined in the data model. This enabled flexible storage and finely detailed and speedy search. Database systems were originally designed for storage on peripheral devices such as rotating discs. In time, the hugely increased size of memory banks available made it feasible to store small databases within the computer processor's directly connected memory store, accessible there by program. Small-scale and mass-produced cassette tapes and floppy discs were introduced to store programs and data for early microcomputers, using simple filing systems. These rotated slowly but were robust and skilfully engineered with smaller disc read/write heads. In time, fast-spinning micro disc drives and solid-state storage devices became available for microcomputers, in parallel and keeping pace with the increasing sophistication of their operating systems. The interplay of program and disc access had by now clearly departed way beyond the long-winded and slowly winding capabilities of early magnetic tape media.

Standardized methods arrived for querying data within relational databases (Structured Query Language, SQL), and this had a profound impact on the industry. The database software market developed rapidly and was a highly competitive one. Keeping pace, as a user, was hard. Products came and went. IBM's database management system (DBMS) blossomed early and died away; Oracle stayed the course and reaped rich rewards. Ingres metamorphosed into the widely used and open-source PostgreSQL database. Document-oriented databases proved widely applicable, still, and MongoDB scaled to huge numbers of downloads. Web applications led to Resource Description Framework (RDF) as a serialization format for data exchange, and SPARQL as a corresponding query language. Extended Markup Language (XML) emerged from Structured Generalized Markup Language (SGML) as a widely used, easily read and machine processable, flexible markup language, to accommodate the transfer of a wide range of structured data in web environments. JavaScript Object Notation (JSON), likewise, and the Representational State Transfer (REST) standard provided Application Programming Interfaces (APIs) to manipulate XML representations of data in web resources. Each packet of data transferred

was understood independently of any preceding transfer and state of the system, the process thus described as 'stateless'.

Capacity and performance requirements became more demanding as databases spread more widely into industry and commerce. As complexity and volumes of data expanded, the engineers enhanced the functionality of their products by providing shortcuts in the processing required, within the database, and implementing more efficient storage, retrieval and querying operations. Intractable problems arose in the early era of implementation of these very large databases–what was termed 'Big Data'. These challenged the applicability and scalability of the relational model of data. 'Non-relational data models' re-emerged, to cope with the new requirements for accessing large aggregations of less well-structured data.

The explosion of scientific measurement, described in Chapter Three, and the emergence of Internet and web resources over the passing decades brought new priority to the storage and processing of largely unstructured data, or data too complex, diverse and largescale to be managed efficiently within the relational paradigm. Free text, medical images, geographical maps and observations and measurements from almost every domain were being collected in ever greater amounts, and users needed to search and analyze them in novel ways. This led to experiments with new algorithms and data structures. Networks of processors and data storage devices led to experimentation with new operating system paradigms, dividing up and implementing processing tasks across multiple machines and storage devices, located both close-by and at a distance.

This period led to the rediscovery of earlier pioneering non-relational database paradigms, notably those based on the storage of 'key-value pairs'. This was a push for greater simplicity, resembling that which had motivated the MUMPS community many decades before, in the late 1960s. The territory first explored by MUMPS was revisited. Packages focused on management of 'big data', such as Apache Spark and Apache Cassandra, originated in academic departments in America and were freely licensed, open-source. Apache Hadoop, also made available under open-source license, tackled the combined challenge of scale of data and computational load. These new systems were described as 'no-SQL' databases and, interestingly, computer languages used to program them revisited another software paradigm from the 1960s and 1970s, that of 'functional programming', which I discuss in the section below on software.

The compromises required to be made between ideas and ideals for modelling data structure and storing it physically, to support rapid and efficient data processing, have always been stark and limiting ones, dependent on the properties of the data, device and system concerned, and the purposes served. Experience of experimental methods deployed to

organize data and the context of its creation, helped to elucidate theories of data model, that in turn guided and harmonized the underlying data and metadata models used for implementations–a virtuous circle of theory and experiment, but, in practice, a slow, tortuous and costly one. This evolutionary process led to clearer language for description of the types and classifications of data, and their aggregations. The wish to describe and formalize the purposes served by the data, and actions based upon it within its user communities brought new challenges into focus, and another era of experimentation evolved.

Information Model

Thus far in the village data processing analogy, we have moved from village documents and lists to village databases. We now move from the village to the nearby town and its organizations and businesses. Here reside developers of systems, makers and sellers of products and services, and those who manage and regulate them. Connections ramify widely with other people and organizations, as collaborators, suppliers, customers and competitors.

The data arising in these wider contexts, and how they are used, reflect wider purposes that draw on and contribute to the evolving knowledge of the people and organizations involved–describing how, and how well, they function. A full circle of meanings answering to questions that might be asked of the data come into play–the who did what, when, where, how and why questions about what it represents and the context in which it arises. This is the domain of semantics. Put simply, what are the meanings that the data represent? This wider scope extends the design of information systems into new areas of modelling of data, covering the processes involved and the outcomes achieved in its capture and use. It also introduces requirements for describing and linking between different kinds of data and databases, and into the world of knowledge bases.

To begin with, additional semantics were layered into systems by way of programs that interrogated existing databases and organized them to serve the additional purposes that exceeded the functionality provided by those sources alone. The market was looking for ways to generalize the specifications for such systems so that a design might be implemented on more than one database provider's product. Here emerged the next stage of abstraction in the design of systems, setting out in more detail, at a higher level, the definitions, interrelationships and flow of data within a system or organization.

This came to be known as an 'information model'. From the mid-1990s, pioneers of database theory, and the companies in which they worked, collaborated to envisage and design an abstract modelling language to express these new requirements. This built on the 'object-oriented software' paradigm, whereby the classes of relational database theory were represented as objects in information models. These objects described both the data themselves and incorporated program methods to manipulate them, stored within the database.

This idea, and that of 'inheritance' of data attributes between data objects in a hierarchical arrangement–a distinguishing feature of object orientation–stretched modelling beyond the focus and potential of relational database theory. Mapping from one to the other presented computational difficulties and imposed limitations on what could be implemented in practice.

As with models in general, the information model is designed to meet the purposes it serves. Just as the relational database model provided a language and mathematically sound underpinning for the design of compliant systems, the information model required its own language theory. Over the following two decades, through multiple incarnations, Unified Modelling Language (UML) became a dominant formalism for expressing information models. It provided a metamodeling framework for all such models. It is not without its detractors who highlight requirements that it cannot satisfy. As ever, the particular case solution veers towards a bespoke, all or nothing approach, and the general case solution counsels an 80:20 perspective of wider applicability. They are not right or wrong choices–they are empirical judgements, and the proof of the pudding is in the eating.

The information model is a further stepping-stone in the design of a database. The data persisted, conformant with the information model, enables application software to cater more straightforwardly for the needs of all users of the data, in their different user and organizational contexts. This database will still require all the previous data modelling capabilities that underpin the way data are stored, searched and retrieved from physical devices.

Seen within the context of a spectrum that spans from data representation to knowledge representation, an information model lies intermediate between data model and its associated data storage schema, and knowledge model and its associated formal methods of inference. It represents a middle layer of semantically enhanced data and knowledge customized to support action. As rehearsed in Chapter Two, knowledge expressed as information with causative power (Deutsch) and information as knowledge for the purpose of taking effective action (West Churchman and Ackoff), appears to be grasping at much the same philosophical nettle. When building

everyday computer systems, we should best avoid descending too far into philosophical rabbit holes!

Information models can in principle be mapped to and persisted within different paradigms of database. In the Internet age, the focus has shifted to less formally structured data, modelled with tools such as the XML markup language. Databases have exhibited a mixture of model paradigms–native XML, key-value stores of the MUMPS kind, as well as relational.

Knowledge Management

Chapter Two drew on histories of philosophy, mathematics and logic in its discussion of knowledge, and reasoning with knowledge. Chapter Three placed knowledge in the context of theory and experiment in the sciences, and the 'omnuscular' world of data that they embody. Chapter Four extended knowledge and data into the world of modelling and simulation using computers. This chapter pursues a further angle on these matters–that of the design of computer systems to represent, communicate and reason with knowledge and data in their practical contexts.[33] In the mid-1950s, the use of computer software to reason in this way became a curiosity and subject of experiment. The exploratory software systems developed came to be called 'expert systems' and a prominent seedbed of this movement of several decades was Stanford University in California. The team and developments there were early forerunners of knowledge-based systems and artificial intelligence, today. The history of these endeavours is interesting, and I latched onto it as I started to read my way into computer science and information engineering in the late 1960s.

I recall fascinating work by the electrical engineer and computer scientist, Ivan Sutherland, that caught my mind at the time. In 1988, Sutherland received the prestigious Alan Turing Award, for his pioneering and visionary contributions to computer graphics. His 1963 Sketchpad software set the scene for human/computer graphical interface. The paper set out two connected problems concerning the readily tangible example of the arrangement of a system comprising three oblong wooden blocks. He showed the complexity of writing a program that would receive as input an arbitrary set of coordinates defining a three-dimensional grouping of these blocks, represent their arrangement as a data structure, and then answer the question: is this an arch? This work addressed a problem of representation

33 As mentioned before, there is some overlap and repetition in the text as I connect overlapping endeavours, chapter by chapter. This is to serve the interests of open-access online publication of the book, where individual chapters are framed as separately downloadable components.

and reasoning with knowledge. This sort of mental gymnastics was the stuff of punitive coursework in the London Institute Computer Science MSc course, along with rather duller, but equally bemusing, exercises in writing machine code to perform floating point arithmetic! Sutherland's program, tackling a question that a human eye might decide in a blink, albeit with some edge cases, was bespoke to this one geometrical problem and ran to many pages of printout. A thought-provoking juxtaposition!

On the east coast of America, the story of IBM was starting to unfold, and on the west coast, Stanford University was an academic nexus of experiment with the new machines. IBM established its own research centre in New York State. In 1956, just a few years after Francis Crick (1916–2004) and James Watson experienced their double helix Eureka moment, Arthur Samuel (1901–90)–a latter-day game-playing Demis Hassabis–who coined the term 'machine learning', wrote a program for an early IBM 704 computer, to play the game of checkers (draughts, in England). This played the game and learned from its experience to play it better, just as DeepMind's AlphaGo has learned to be the best at Go and AlphaFold to be capable of folding proteins.

In the 1950s, IBM Checkers was the nucleus of an excited explosion of interest and litany of terms: expert system, knowledge representation, knowledge management, knowledge engineering, machine learning and artificial intelligence took flight. Discipline struggling for identity tends to struggle over its name–my own a foremost culprit! Writing about this confusion takes too much time, has too much imprint and has little meaning, other than as struggle for a yet unreachable clarity. The volume of words is noise more than signal, reflecting that we do not yet, and maybe still cannot, understand. In this buzz, all manner of flowchart, decision tree, statistical transformation and program logic have been claimed to have a place in the oeuvres of the day. The unfolding of the story of the encounter between computer and knowledge is a still-evolving story of insight and understanding, all around the circles of knowledge and of available computational theories and methods.

In this chapter, the focus has been on engineering. The information model lay in the middle, between the quest to model and engineer data and the quest to model and engineer knowledge. Knowledge representation and access to and reasoning with knowledge are in themselves lofty goals. The next step of integrating a knowledge model descriptive of how a system works, with a database of records collected from the working system, is still a highly experimental challenge. In 1950, Escher made a lithograph illustrating the contrast of imagined and perceived order in a messy world,

entitled *Contrast* (*Order and Chaos*).[34] The lithograph is a telling metaphor for how we speak of computer systems and methods, and what they are often made up of, under the bonnet!

Experiments involve trial and error, and the world-weary aphorism about experiments in computerization has it that 'to err is human, to really mess things up, buy a computer'. The computer is a hard task master in exposing inadequate or inconsistent thinking. Computer programs are unforgiving in their insistence on clarity and consistency in what they are required to enact.

The term model, as used in the data model for representation and manipulation of data collected from a system and stored in databases, assumes new significance when used in the context of representation and reasoning about the structure and function of the system itself. This model is informed by a mixture of current understanding and ideas about the system, and data collected from it, historically and prospectively. Here, concern for integrity of data widens into concern also for correctness of reasoning about the structure and function of the system itself. Representation of knowledge and reasoning draws on theory informed by philosophy, logic, linguistics and mathematics, and computational method rooted in changing science and engineering. Data are not just things we need to acquire and record. They have a wider context, representing our quest for knowledge and classification–exemplifying our understandings and ideas about the nature of things and the organization of our knowledge about them. 'Thing theory' became a thing–occupying abstract and philosophical minds! It is a bit like string theory. I am eagerly awaiting strings of things theory!

There are multiple ways in which knowledge and reasoning draw on data and make logical inferences, far exceeding the making, manipulating and optimizing of lists. Knowledge engineering describes all kinds of description of reasoning about things. Things may be kinds and parts of other things. A Ford car is a kind of car; a Ford car wheel is a kind of wheel and a part of a Ford car. 'Kind of' and 'part of' are relationships. These can be pieced together in trees or networks of relations, in like manner that we saw with groupings of data. Logical propositions drawing on these relations may be true or false, and theorems based on them may be proved logically correct or incorrect.

As with the proliferation of connections made in the early experimental exploration of data models, this field quickly also connected very widely. The language of mathematics and formal logic is the proper and safe place in which to discuss theory underpinning data and knowledge representation.

34 M. C. Escher, 'Contrast (Order and Chaos)', *National Gallery of Art*, https://www.nga.gov/collection/art-object-page.63273.html

I have touched the surface of this discourse in a limited and general manner. Further curiosity is best pursued into specialist literature. It is a difficult domain to express, communicate and interpret. It is often expressed confusingly. What can one make of the self-reference in a definition of relation as 'a set of concepts and categories in a subject area or domain that shows their properties and the relations between them'? Having dumbed down on meaning, it is just then a short hop to statements like 'what's new about our ontology is that it is created automatically from large datasets'!

Knowledge bases have a two-fold purpose: capture of the knowledge relevant to a domain of interest, in some computable form, and formal and computable method for reasoning with that knowledge, in the context, also, of new and prospective data collected in the domain. The representation of knowledge in this way is reasonably termed a knowledge model, although all forms of knowledge might also reasonably be said to model reality, in one way or another. As we saw above, the earlier term used to describe systems of this kind was 'expert system', deriving from early attempts from the 1960s, to represent complex human decision making. Some early systems involved simple programs to enact the logic of a decision tree. Others showed ground-breaking potential. I introduced some early examples of knowledge-based systems in Chapter Two, in the discussion of formal logic.

Having tracked the American story, another story connects this new world back to the English genius of Turing and Donald Michie (1923–2007), a luminary founder of artificial intelligence (AI) in the UK. His expertise spanned an Oxford degree in classics, wartime contributions in code breaking at Bletchley Park, alongside Turing, an MA in human anatomy and physiology, and then a DPhil in mammalian genetics. Long classified as secret, Michie's work on using the Colossus computer to help decode messages from a German encryption device, more sophisticated than ENIGMA and nicknamed Tunny (the encrypted messages were called 'fish'!), was fundamental. Michie and Turing shared a hobby in programming computers to play chess and were captivated by the idea of machines that might learn from experience.[35]

In his next appointment within the medical school in Edinburgh, Michie co-wrote an early textbook of molecular biology and then, from 1960, switched his attention back to AI. He created Freddy II, the world's first robot that could work from computer vision to assemble complex objects

35 The team at Bletchley Park included the three times British chess champion, Harry Golombek (1911–95), as I discovered at a recent visit there. Members of the team wrote a letter pleading for more funding for the code-breaking effort. Golombek took this to Winston Churchill, who responded within twenty-four hours, issuing instruction for 'action this day' to provide the funding!

from a pile of parts. His wide-ranging and speculative research interests disturbed his academic chiefs and, in 1973, the UK Science Research Council commissioned James Lighthill (1924–98) to report on the prospects for AI. I knew him as the then Provost of UCL, where I was at the time engaged on my PhD. The dismissively critical Lighthill Report derailed robotics research in the UK and wider afield. Another example of 'Airy' dismissal of new ideas that has cropped up several times in these pages–in this case, more aristocratic academic hubris than pretence of knowledge! Not so much losing the plot as just not finding it in the first place![36] Japan did not make this mistake and its industrial developments in robotics of the 1980s helped power a burgeoning economy. The Japanese progress towards '5th Generation Computing' and AI led, fearful of competition, to a revival of interest in robotics in Europe and the USA. In the mid-1980s, Michie headed the Turing Trust in Cambridge and established the Glasgow Turing Institute, where he pursued work on robotics, machine intelligence and computer vision into his retirement in the 1990s. Among his later interests were microcomputer systems dedicated to surgical audit and patient administration, word-processing and spreadsheets.

The previously described Escher lithograph, *Drawing Hands*, might be taken to symbolize information as a self-referencing complementarity of knowledge and data–information as knowledge from data and data from knowledge, perhaps. It might capture the complementarity of an engineer making and doing something and knowing how to make and do it. It might symbolize informatics as a co-evolving science and engineering of information. To continue in this way of looking at information engineering, we must now consider the algorithms and software that specify the instructions that the computer enacts. This then leads on to the design of information systems and efforts to promote their coherence and mutual compatibility through information standards and standardization. The final section considers the profound changes that the Internet and World Wide Web have brought to information engineering.

36 It is both interesting and slightly ironic that Demis Hassabis emerged from a PhD combining computer science and neuroscience at UCL, the institution where James Lighthill had been Provost, and took his world-beating skill in the games of chess and Go into computer games and then into DeepMind. Also, that Go, the hardest of board games, should have a name which, in life science, is the acronym for Gene Ontology, and that, in this domain, the DeepMind AlphaFold software looks to be mastering the highly complex three-dimensional combinatorial problem of life science, that of protein folding.

Software–Algorithm, Data Structure and Computer Program

The story and timeline of the evolution of computer software has been convoluted and long, as have been those of data model and database, and knowledge model and knowledge base. Wirth wrote in the mid-1970s, that 'Algorithms + Data Structures = Programs'. Algorithm is an abstract concept used to describe a method for manipulating data in a computation. Programs enact computation. There has been a continuous tug-of-war and coevolution on these three fronts, under countervailing pressures of theory and practice. Let us start with program as language.

Many different programming languages have emerged, traditionally grouped within four principal programming paradigms, although there are admixtures of these. Procedural, object-oriented, functional and logical paradigms constrain the ways in which a program can be expressed, based principally on abstractions of process, data, algorithm and logic. Each programming language provides a distinctive repertoire of methods and styles for the programmer to use in expressing and representing the algorithms and data required to implement their desired computation. Programming languages have evolved to address requirements arising in different application domains, and learning acquired through experience in their use there. It has been a chaotic era–new technologies and capabilities arriving in rapid succession, onto a landscape of then current practice rendered quickly obsolete. The story continues to unfold. Quantum computation is a window opening onto another new world–or multiple worlds!

The combined mathematical and computer science foundations of today's programming languages were set in the first half of the twentieth century, notably in the 1930s by Turing's universal computing machine and Alonzo Church's (1903–95) lambda calculus. These provided a mathematical framework for formalizing the language of computation, rather as Frege's predicate calculus had cast a mathematical net over the language of logic. And, as introduced in earlier sections of the book, the first electronic computers evolved in the early 1950s, from wartime prototypes in the USA and England–the ENIAC with its connection to the mathematician von Neumann and weapons research at Los Alamos, and the Colossus with its connection to code breaking endeavours at Bletchley Park in the UK. Early programmable computers implemented a generic design of a computing machine (central processing unit or CPU), known as the von Neumann architecture. Programmed instructions for each machine were expressed as

a numeric machine code and loaded into the computer memory along with data, to instruct the machine to perform the calculations required.

Early programs were expressed directly in these numbers. Programmers became fluent in the language of machine code, often expressed as a sequence of hexadecimal numbers, one hex number for each four bits (binary digits) of machine code. Early symbolic languages arose in the 1950s, in the form of assembly languages (assemblers) that translated from symbols representing instructions for the processor to execute, and locations within the CPU registers or computer memory, where program and data were to be located. Machine and assembly language expression focused on the inner workings of the computer rather than on the structure of the computation that the program was designed to perform.

Programs at this level were able to exert a fine level of control over machine operation and program execution. They could easily contain erroneous code that halted or otherwise crashed the operation of the machine. Such programs were opaque and cumbersome to correct or 'debug'. In early days this often involved the flicking of switches on the computer console, to inch, one step at a time, through the program instructions and observe the binary contents of machine registers and memory displayed on panels of console lights. I've done my share of puzzling over these twelve- and sixteen-bit binary numbers and their hex-codes, including times when the lights were intermittently faulty! You can imagine the emoji-like frustration–but it did teach one about what was happening beneath the bonnet of the machine and help keep one's feet grounded!

High-level programming languages were experimented with to express the steps required in a computation, independently of the features and vagaries of the machine on which it was to be performed. The design of these early languages reflected the kinds of computation their designers had in mind–FORTRAN focused on numerical calculation and LISP on symbolic reasoning tasks, for example. The needs of compact and expressive data structure and algorithm, and efficiently performant programs, might easily have pulled in different directions, albeit that they all, necessarily, interacted in execution of the task at hand.

From these beginnings, numerous clans emerged, championing many threads of imperative programming and declarative programming languages. Imperative programming focuses on describing *how* a program operates. Declarative programming focuses on describing a program's desired results rather than on steps the program is to perform to achieve them. In those embryonic times, FORTRAN was more imperative in style, and LISP more declarative. It was a highly experimental era and theory came later. Over time, two principal paradigms and priesthoods took hold on the landscape of experiment–temples of object-oriented programming,

dedicated to gods and genes of data, and functional programming, dedicated to those of algorithm. Genetic recombinants drew pragmatically from both gene pools. Bespoke languages were created to focus on the characteristics and needs of particular subject domains–domain-specific languages.

Having toured and ramified to the horizons of data-oriented and functionally oriented programming styles (implemented for ensembles of mainframes, minicomputers and microcomputers, on the uncharted and shifting sands of new domains of computation), program language and software discipline are now drawing together under the unifying hardware technology umbrella of the Cloud and the software and network technology ecosystem of the World Wide Web. The history of language is that it evolves and ramifies, unifies, breaks and regroups. It is inevitable that we have not seen the end of new paradigms and languages of computation!

Maybe someday there will be a priesthood of all software believers, but one must rather doubt it, as science and technology continue to evolve. Maybe the music of all software will one day be tuned to the key of F-sharp Major (F# is the name of a favoured modern functional programming language!)[37] It could have Mahler's Tenth Symphony as its theme music, but, then again, that was unfinished! The challenges of writing and transcribing music for the different instruments in a full orchestra will forever characterize the quest for a universal language for programming, and indeed for mathematics. All theories, and florid analogies, like this one, break down somewhere! There has been a Mozart Programming system since 1991, though, and the pursuit of harmony is a good goal, countering countervailing pressures that tolerate or actively seek unnecessary and undesirable division![38] But I should not be too sceptical and dismissive–there has been a lot to admire in what this often-chaotic era has achieved, in evolving, clarifying and tidying the world of computer programming, for those who enter seriously and practise it today.

Moving on now to algorithm, the term itself has an ancient pedigree connected with the mathematics of algebra. Algebra dates from the early

37 F-sharp major is the key of the minuet in Haydn's 'Farewell' Symphony, of Beethoven's Piano Sonata No. 24, Op. 78, of Chopin's Barcarolle, of Verdi's 'Va, pensiero' from *Nabucco*, of Liszt's Hungarian Rhapsody No. 2, of Mahler's unfinished Tenth Symphony, of Korngold's Symphony Op. 40, of Scriabin's Piano Sonata No. 4. Wikipedia contributors, 'F-sharp Major', *Wikipedia, The Free Encyclopedia* (6 July 2023), https://en.wikipedia.org/wiki/F-sharp_major

38 'The Mozart Programming System combines ongoing research in programming language design and implementation, constraint logic programming, distributed computing, and human-computer interfaces. Mozart implements the Oz language and provides both expressive power and advanced functionality'. *The Mozart Programming System*, http://mozart2.org/

recorded history of the Middle East, in the arithmetic of number and the use of symbols and words to represent and reason with numbers, in reaching solutions of equations. The word is traced to the Arabic title of a treatise of the seventh-century Persian mathematician Muhammad ibn Mūsā al-Khwārizmī (780 CE–850 CE) on the solution of linear and quadratic equations. He is credited as the father of algebra. Later, Latin translations of his work transcribed his name as Algorithm. The meaning and usage of this term has evolved also from Greek αριθμός [*arithmos*] ('number'), Medieval Latin 'algorismus', Middle English 'algorism' to 'algorithm' in seventeenth-century English. It has become specialized within the language of mathematics and computer science. For example: 'In mathematics and computer science, an algorithm is a finite sequence of well-defined, computer-implementable instructions, typically to solve a class of problems or to perform a computation';[39] or, more generally, 'a process or set of rules to be followed in calculations or other problem-solving operations'.[40]

As with the history of mathematics, that of the algorithm is a story of coevolution of theory and practice. What has happened over millennia in mathematics has occurred in a century of coevolution of mathematics and computer science, and more recently in five decades of Internet time. It is a daunting field to seek to summarize here. I take courage from a Faraday Lecture at the Institute of Electrical Engineering in London, some fifteen years ago, on this subject. It was delivered by Donald Knuth, the American computer scientist and mathematician. Hopefully, he will be remembered, millennia from now, as the father of the analysis of algorithms–maybe they will then be called knuthisms! Knuth had drawn from the practice into the theory of algorithm–how programs systematize computation. As I started my journey into computer science in 1969, Knuth was already publishing his seminal encyclopaedic volumes on *The Art of Computer Programming*,

39 Wikipedia contributors, 'Algorithm', *Wikipedia, The Free Encyclopedia* (1 July 2023), https://en.wikipedia.org/wiki/Algorithm

40 'algorithm', in *Concise Oxford English Dictionary*, ed. A. Stevenson and M. Waite (Oxford: Oxford University Press, 2011), p. 31. In like manner, mathematics has evolved in the form of number theory, geometry (measuring the earth) and analysis, and in modern times through the study of combinatorics, probability, statistics and numerical methods. The monumental *Wiley Handbook of Applicable Mathematics*, published from 1980–1984, with a supplement in 1990, has volumes on Algebra, Probability, Numerical Methods, Analysis, Combinatorics and Geometry (parts A and B) and Statistics (Parts A and B). With my McMaster colleague Ralph Bloch, and halfway along my songline, still immersed in mathematical modelling of human physiology, I assembled and edited a companion handbook in the series, entitled *Mathematical Methods in Medicine*. This was published in two parts, in 1984 and 1986, covering statistical and analytical techniques and clinical applications, respectively.

which he started in 1962, envisaging a seven-volume series.[41] By 1973, the first three were in print from his handwritten text. Part one of volume four (IVA) took another thirty years and part two of volume four (IVB) was published in 2022. Along the way, the typesetting technology used to set his elaborate notation became obsolete and he spent eight years developing TeX, to enable publication of the fourth volume to progress.

After this build up, you can imagine the keen anticipation of those attending the lecture. Except that a lecture did not take place. The huge auditorium was packed out for this annual event commemorating the scientist who pioneered electromagnetism and electrochemistry, Michael Faraday (1791–1867). The hall hushed and Knuth, by then quite elderly, came slowly to the lectern, said one sentence, and sat down. 'I will take questions', he said! Slightly startled at first, the audience revived to pump him with questions, and he to respond thoughtfully, for the allotted hour. It was a memorable occasion and quite interesting as a reflection on the way one, such as he, oversaw the almost limitless potential scope of their topic.

An algorithm is an abstract representation of steps taken in computing with numbers and symbols, organized into useful and tractable data structures and operated on with useful and powerful functions. It can be expressed and implemented in different ways, employing different kinds of computer language, running on different kinds of computing machinery and serving different kinds of purpose. How programmers choose to pick their way through this combinatorial set of options and opportunities is up to them. Thereby, their program may or may not work, be efficient to code and execute, and reach a correct answer in line with the question asked. As with Knuth's audience of programmers, what did they need and want to know about algorithms, which they could draw on in their programs and deploy.

Algorithms are manifested and used throughout academic and professional discourse. This summer (2020) they have guided, and then been derided, in the context of the grading of student performance, when national school-leaver examinations were cancelled. They are sometimes slow and laborious to articulate and program. An arbitrary list of numbers might contain different types and properties of numbers–for example: integer or real, odd or even. They might contain prime numbers. The

41 Completed volumes are as follows: Volume I: Fundamental Algorithms; Volume II: Seminumerical Algorithms; Volume III: Sorting and Searching; Volume IVA: Combinatorial Algorithms, Part 1. Planned volumes are as follows: Volume 4C, 4D: Combinatorial Algorithms, Part 3 and 4; Volume V: Syntactic Algorithms; Volume VI: The Theory of Context-Free Languages; Volume VII: Compiler Techniques.

programmer might wish to sort and order the list in different ways. Given a representation in computer memory, using what methods can the list be sorted, to be presented as a sequence ordered by size of number, further ordered into two groups of integers and reals, or two groups of prime and non-prime numbers?

Algorithms for achieving desired ends, such as these, might, in principle, be enacted in the head, or with pencil and paper, or by implementation through a computer program. There may be a variety of potential methods that come to mind; some easy to envisage and others more brain-aching. Some are easy to enact mentally and some too hard or time and resource consuming, that way. The first two of the above orders are mentally tractable, and practical to achieve, depending on perseverance and length of list. Prime numbers get more complicated to reason with, both mathematically and mentally. New theorems about prime numbers gain Fields Medals in mathematics. Contemporary cryptography depends on the identification of the large prime numbers it is based on, being computationally intractable.

Some algorithms for achieving a desired end may be correctly described in a concise conceptual form, and others equally correctly, but more laboriously. This may reflect the intrinsic nature of the task itself, or the expressiveness of the language used in expressing, reasoning with, and programming it. Equally correct algorithms and programs may vary hugely in their comprehensibility to a human reader trying to understand them, and in the efficiency of their execution by the computer, which likewise may reflect the characteristics of its hardware and operating software. Proof of the correctness of an algorithm is a matter of mathematics and logic. Mathematical and logical proofs are expressed in mathematical and logical languages and notations.

Computer programs enact algorithms. Program is written in programming language and its correctness is partly a matter of the correct use of that language and partly its correct implementation within the computer. As Knuth is quoted as saying, 'Beware of bugs in the above code; I have only proved it correct, not tried it'[42] and 'An algorithm must be seen to be believed'.[43] Good program design reflects a match and balance between the form of a particular algorithm and how it can be expressed in a particular programming language and computing environment. These are attributes of the language, the numbers, texts and symbols of its datatypes, and the methods whereby it accesses, manipulates and stores these in

42 From 'Notes on the Van Emde Boas Construction of Priority Deques: An Instructive Use of Recursion' (1977), n.p.

43 D. E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, 3rd ed. (Reading, MA: Addison-Wesley Professional, 1997), p. 4.

enacting the algorithm. These considerations have reflected in the different programming paradigms that have evolved over time, in turn reflecting the problems or tasks being computerized, the data structures being processed and the computer systems in which programs operate and interoperate with others.

There is a framework and hierarchy of good order in all this–traversing theory and practice of mathematics, logic, algorithm, data, program and machine–on which we depend so that the computation tackled by the program can be relied on to do what the programmer and user want it to do. This framework requires discipline grounded in all these domains, both one at a time, and taken as a whole. Practically, there are differences in assumptions made and approximations implicit in different parts of the system deployed. There are impedances to free flow of information between them–a so called impedance mismatch, drawing on analogy with electrical circuit design. Philosophically, there are differences of belief and perspective about reasoning.

In 1962, Knuth was a graduate student in mathematics at California Institute of Technology (Caltech), close by to Richard Feynman (1918–88). At the time, he conceived of his project as a single book in twelve chapters. The sceptical publisher sought his academic adviser's support in deciding whether to take it on. The project has still only reached halfway in shining light to illuminate an expanding universe of algorithm. The light cannot reach the boundaries. It is the ultimate missed publisher's deadline! James Lovelock wrote *Novacene* at age one hundred, so there is still hope that Knuth's volumes on syntactic algorithms, context-free languages and compiler techniques may yet see light of day!

Software is everywhere; it conditions and reflects everything it touches. It is approach and paradigm, art and creativity, experiment and interface, ecosystem and legacy. Faced with a boundless challenge of encapsulating all this, my approach in this book has been to place the co-evolving computer machines and programming paradigms of each era in the context of my career in health care information systems, and the projects I designed, wrote programs for, led, collaborated with and reviewed, over those years. In this chapter and Chapter Seven, I tell these stories, indexed to both timelines.

Software as Art and Creativity

Painting and drawing, sculpture, music, theatre and dance are creative arts. Reasoning with abstract mathematics is a creative art. The arts fire imagination, stimulating and satisfying feeling and emotion. Those who know and practise the arts bring knowledge, skill and experience to its

interpretation and appreciation. They have insight, talent and taste, and they have preference. Writing poetry is an art and so is writing computer programs.

My second cousin and her husband are art gallery junkies, and they take me along to exhibitions and educate me. Jim is a textile designer, who trained in art school alongside David Hockney. His appreciations and preferences in colour, texture and design are a great resource to draw on in interpreting art, such as when we viewed the pre-Raphaelite decorated rooms of the Edward Burne-Jones (1833–98) house on the bank of the river Thames, in London. Skills of artists and designers, and properties of tools and materials, blended and intermingled creatively.

Mathematics, science and engineering are creative, in similar and different ways. My late sister's family are talented mechanical engineers in their work and hobbies, building model engines and restoring and maintaining vintage motor bikes and cars, displaying them at fairs and competing in races. They make and maintain things and derive great pleasure in these skills and pursuits, albeit demanding and sometimes frustrating. In *Zen and the Art of Motorcycle Maintenance*, Richard Pirsig (1928–2013) mused on the engineering of motor bikes and the meaning of quality of machines.[44] Another inuksuk book of mine.

It is not difficult for someone observing from the outside to recognize quality in a practical skill and achievement on view, and the joy it brings to its practitioners. It would be a rare person who visited the experimental nuclear fusion reactor at the Culham Laboratory near Oxford, and heard its story, who did not come away feeling mightily impressed. It speaks for itself, we say. It is more difficult, though, to appreciate abstract worlds of mathematics and computation in the same way. Their qualities and pleasures are experienced within the domain, but not readily beyond it, where they may easily be unappreciated, taken for granted or evoke opposite feelings, especially when they fail, or fail to connect–brain box, nerd etc.!

Although most can rise to playing a hand in cards and board games, excellence in chess or Go verges on an artform. What, then, would Pirsig have made of AlphaGo, I wonder? In *The Creativity Code*, Marcus du Sautoy has written memorably about artificial intelligence and creative art.[45] I have used the 2022 released Stable Diffusion artificial intelligence-based software to create images from text (see Figure 1.1), to illustrate how new artforms are coming into existence in the Information Age.

44 R. M. Pirsig, *Zen and the Art of Motorcycle Maintenance: An Inquiry into Values* (London: Bodley Head, 1974).

45 M. du Sautoy, *The Creativity Code: How AI Is Learning to Write, Paint and Think* (Cambridge, MA: Harvard University Press, 2019).

These parallels of software with art and creativity occurred to me recently, when observing the vivid and detailed communications online, worldwide, minute by minute, among enthusiasts keeping alive the hardware and software of vintage computer games. My younger son, Tom, now a cardiologist, had told me a week before about the open-source software created to emulate the now obsolete MS-DOS microcomputer disk operating system, on which Microsoft was built forty or more years ago. He had used it to show his children the early computer games and exploratory worlds he played with in his teenage years. It occurred to me that my long-forgotten copies of the computer simulation programs, the Mac Series, that I described in Chapter Four, might be coaxed into running again on this open-source DOSBox platform.

I collected the 1987 floppy discs that we published with IRL Press, which are, as far as I know, the sole surviving implementations, and bought a floppy disc drive to try to read them again. This arrived the next day and I succeeded in reading the discs onto my backup Windows-XP laptop, not wanting to take any unforeseen risks in loading them onto my current machines. The appearance of the initial screen of the simplest of the programs made my heart jump, only to be immediately disappointed as the software crashed as soon as the first user input was requested from the keyboard. After digging around in the online manual, I adjusted the configuration of the DOSBox software but achieved only minor correction of the opening program graphics–the keyboard crash persisted. I joined the online community and scanned its extensive lists to try and find reference to the problem I had encountered, but to no avail. Responding to my enquiry to the DOSBox development team, a very helpful person in the Netherlands got in touch immediately. After hearing my description and seeing the details of the crash from the screen dumps that I emailed to him, he came back with a patch that instantly solved my problem. All four programs sprang to life. He told me in subsequent emails about the DOSBox open-source community he worked in, as a hobby, with colleagues around the world.

In this process, I saw a humming beehive of communication among games afficionados, maintaining a honeycomb of long-extinct computer hardware and software for running a huge range of computer games. This was vintage motorbike community writ large in vintage computer games community. Joyful hobbyists, wielding soldering irons, not standing at lathes; tuning software set up, not engine timing. And competing in virtual reality, online, not in driving spluttering vehicles up muddy hillsides!

Computer programming involves theory, skill, practice and creativity. It has evolved from and illuminated theoretical foundations of mathematics and computer science of the past one hundred years and more. The practical capabilities of computers and software have balanced with the elucidation

of theory, both for the methods used and applications devised. The field has advanced on an Internet timescale, rendering hard-won skills and achievements quickly obsolete, creating a Whitehead socio-technical anarchy of legacy and confusion. Theory and practice of computation, and its devices, software and systems, continue to co-evolve. The art and creativity of programming is expressed with a continuously evolving palette of algorithm and language on an ever-growing landscape of applications.

Software as Experiment

It is quite rare for non-programmers to write good code, but some do create innovative software for their own domains of specialism, with impact that it is hard for those versed only in programming to improve on. Some of the most original software I have studied was developed by people who had the inspiration to experiment with use of the computer in a creative way, in a new domain. Many have been talented and polymath scientists and clinicians–John Dickinson, Arthur Guyton, Bill Aylward, Sam Heard, Octo Barnett. Some of their code reflected limited finesse in engineering and design, but it had the merit of a clear goal for how the program was intended and needed to fit and work within its planned context of use. They equipped themselves with sufficient skill to bind the computer to their will, working within its features and around its limitations, to achieve the purposes they had in mind. The code they produced was sometimes a tangle of data and algorithm. Their creative triumph was to express their huge knowledge and insight in physiology and medicine in a computable form. Their work broke new ground across both scientific and technological domains. They made connections. In doing so they were, in themselves, inuksuks in the landscape of software advance.

These people were architects more than designers of the programs they produced. Architects must learn and understand, by iterative experiment, how their planned work will fit and function within its intended setting. They must bridge from the user of the system to the science and technology embodied in its software and hardware. Lacking the expertise provided by bridging insight and experience, the construction team building a bridge may fall into the sea below. Of course, pioneers sometimes fall from their novel bridges into the same sea. But they tend to be strong, and know how to swim there, then get out of the water and go back doggedly to build a better bridge; or know when to abandon the effort!

This analogy with bridge-building came to mind when recalling the Eiffel bridges that I was once told about, in Porto in Portugal, during a European Union (EU) project team meeting there. Eiffel designed novel and

beautiful bridges as well as the iconic Eiffel tower in Paris and the American Statue of Liberty. His creative vision led him to take risks in experimenting with new construction methods and materials. There was sometimes doubt that they were sound, and the first unshuttering of the new structure was a public spectacle, with the audience assembled wondering whether the new design would collapse! Even fully accredited engineers make mistakes in the design and oversight of their constructions, as the wobbling and buckling London Thames and Tacoma bridges have attested.

Software as Struggle with Imperfection

In 1977, I listened to a memorably entertaining keynote lecture delivered at the closing conference of the UK's National Development Programme for Computer-Assisted Learning, at the University of Surrey. The speaker was Judah Schwartz[46] and his title was something like 'How to do maths with a broken calculator'. The talk was about his experiments in teaching maths by challenging his students to calculate using a limited subset of the calculator's keys–the others being effectively 'broken', hence the title. The message was about the nature of fluency in mathematics and how it is taught, and the implications for the teaching profession and education technology. Schwartz showed that much can be taught and learned about the nature of calculation by seeing how one can get along when our methods of calculation are broken, or not available in some way.

There is something of this challenge in the writing of software with 'broken' programming languages–akin to 'broken' formal logic limiting our ability to express nuance of logical argument. All use of language involves struggle to express what the originator wishes to say, and to be heard and understood. All computer systems represent a struggle between the imperfections of what they can and cannot do, and the requirements of the domains in which they are employed. Quite human, really! I wonder how Schwartz might have turned this struggle with imperfection into useful learning about those domains. There's a lot to learn, here, about future health care.

Software development selects and draws on available methods and resources and invents new ones. It homes in on goals–what is to be achieved, where it is to be enacted and how it will be tackled. There is an ever-expanding and evolving variety of programming tools and resources to choose from in

46 Schwartz was a luminary figure in educational technology whose career spanned physics at Tufts University, engineering at Massachusetts Institute of Technology (MIT), and education at Harvard University. He published his 'broken calculator' software for the Apple Mac computer. He died in May 2020, from Covid-19.

tackling these tasks more effectively and efficiently. As creative art, these are akin to knives, brushes and pencils; to canvases, chapel ceilings and street side walls; to oils, charcoal and ceramics. The result may be a Leonardo, a Michaelangelo, a Banksy, a graffiti or a first attempt. They are frequently experimental, with preferences and choices highly contextualized.

In the early days, the available methods for writing and testing software were very limited and rather onerous and unreliable. Akin to vintage cars needing to be coaxed to perform, hands on, with crank handles and fine adjustment of timing of ignition and carburation of fuel-air mix. It was said to me by one prominent early supplier, that most of the code written for a large-scale system, like a hospital information system, was taken up with handling situations arising when the desired program operation went wrong in some way. Computer systems are still far from smooth functioning utilities, that merge unseen and unremarked into the background of life.

One learns language as a child by struggling to express one's growing self with a limited vocabulary of words, and sometimes a florid vocabulary of frustration. Natural language takes a long time to learn well and there are natural linguists and struggling linguists. And older minds struggle more than younger ones. Coders like me learned to write programs in similar manner, with the vocabulary of machine code, assembler language, and a limited choice among higher level program languages. Some of these were implemented as line-by-line language interpretation of the program code and some through a preliminary step of language compilation, to translate the code into a block of essentially machine code that could be fed to and digested by the machine.

Programming languages embrace data representation and algorithm. In their varieties, they evolve and enable fluency tuned to different domains of discourse. They can bring rigour and efficiency to the writing of software, and they can tie the process in Gordian knots of unfathomable complexity, when too elaborate and beyond the feasible skills, available time, or interests of their supposed users. Until you have experimented with framing a problem and solving it with the tools available to you, you cannot learn how to do it well. Doing it well, or better, may involve, using the tool better or using a different tool, or combination of tools. This is how crafts and disciplines interact. And all this struggle for learning costs time and creative effort.

The manipulation of bits and bytes, such as is needed in controlling devices, has remained in the world of machine code, assembler and low-level language, notably C and its descendants–not all judged as improvements! Scientific computation was programmed early on, and still exists, in FORTRAN–labelled as an imperative language, describing a process that does what it says it does, step by step. Focus on improvements in structure

and flow of these programs led Wirth to develop Pascal. There was a long-running struggle to balance imperative and declarative elements in the style of program language. Theory and technology of computation and program language, interacting with formal linguistics and grammar, have co-evolved beyond measure along my songline. There was a pattern of continuous experiment and improvement, characterizing and exploring the principal program paradigms.

These were all struggles to find good and feasible ways to express and communicate about fuzzy and grey areas of development and application of program languages. Early programming languages were invented *de novo* and became the shaky foundations of software because there were no others. Inflexible and massive foundations, and the methods, materials and structures of buildings constructed on them, do not easily survive subsidence or shifting sands. Early programs, too, were rather fragile edifices. Keeping such edifices intact and functional is costly and exhausting work!

In the continuing efforts to tame such challenge of imperfection, the languages and skills of programming have become progressively more specialized and commoditized, within software and computing platforms. This has greatly improved the construction and maintenance of new systems but also accelerated obsolescence of the old. Valuable resources that embody useful learning and capability are no longer supported or supportable by these new paradigms, skills and infrastructures. The languages in which they were framed and communicated are no longer spoken. It is a cruelly wasteful process but very natural from an evolutionary perspective. The alarming feature is that the disruption it entrains is in significant part a global phenomenon. It is a largely unseen, underground threat, with overground consequences. It is not a meteorite leading to global winter or viral pandemic. It is, in this sort of lurid analogy, like a continuing liquefaction and reconsolidation of the foundations underpinning every building.

The world of software platforms has created much new and valuable common ground, but also unleashed unproductive competition between factions competing to enclose parts of this new territory for themselves. Markets and market forces are important but they, too, are always a struggle with imperfection. We are coming to see that new mechanisms of collaboration and regulation are needed, as these markets assume global dimensions, such as those created by software systems operating in the earthly Web and computational Cloud of the Information Age.

Software as Ecosystem and Legacy

Software development has become a massive industry, in which platforms and tools have evolved to keep pace, to make programming more efficient, reliable and automated. Generic and standardized approaches have evolved, honed within changing theoretical, practical and commercial contexts. The common framework of the World Wide Web, disseminating information and linking applications, has been an immense driver of this process in recent decades. Multiple new opportunities and risks have arisen. The competing platforms of IBM, Microsoft, Apple, Google, Amazon and Facebook have acquired immense power. When their products are commoditized as services, through Cloud-based networks, they can lead to monopoly that defies displacement by the innovations of newcomers and imposes unavoidable cost on users who are bereft of viable alternatives. This has brought new commercial and governance pressures that are now operating and impacting on a global scale.

The rapidly changing scope and context of users' requirements brings continuing need for investment in the new applications required to support them. This investment focuses on programming methods to integrate with an existing or new, usually proprietary, implementation platform, to maintain an interoperating ecosystem of applications. Choices are limited and based, in large part, on rules imposed for achieving compatibility with the platform. Very sizeable investments are then vulnerable to industry developments and behaviours that may quickly render the chosen platform no longer cost-effective or efficient.

Within a chosen shared software platform ecosystem, different parts of the user organization, and their different supporting specialist suppliers of software, have different needs, rules and priorities for managing change. They may need to move faster than the supplier of the proprietary platform can adapt and change the product to accommodate their special needs. If they act independently, they may choose and implement software that fragments the coherence and integrity of the local software ecosystem, fall behind and harm themselves. If they do not act, they also fall behind. It is a catch-22. And in such contexts, the implementation, testing and maintenance of needed changes can take much time and prove very costly, or indeed unachievable. They wait for something to break and then pick up the pieces.

The evolving combination of inflexible and dysfunctional software brings dissatisfaction and inertia for users, software providers and markets. Powerful software monopolies–powerful because customers become locked to them–acquire a rigidity and inflexibility that makes customization to local need ever more difficult. The user must bend to the system rather than

the other way around. This is not the stuff of innovation. But it is the stuff of accumulating information entropy and disorder.

It is a natural feature of evolution that the underpinning framework and integrity of the software languages and paradigms are challenged by evolving requirements and experience of them in use. Each generation or paradigm is in a sense broken by purposes for which it is found to be suboptimal, outmoded or otherwise unfit. Oftentimes, opaque and intractable code is kept alive by highly intelligent and capable coders who have struggled to keep it operational, leading to more complex and severe code disasters, when the inevitable collapse happens. Cyber-attacks reveal these vulnerabilities, rather as a viral epidemic reveals vulnerability of health care.

Continual bending of software to overcome problems all too easily results in entropic disorder–spaghetti piles of code threads. Arguably, this is a good time to think afresh and, where possible, to start again, having learned important lessons about purposes, methods and tools of software development. Sometimes this will not be possible, for any number of good and bad reasons. In real life, a patch and move on approach often prevails, or proceeds by default. Failed or faulty code is patched; the unsafe bridge is reinforced, and this increasingly unsound legacy builds further towards future collapse.

This is a gloomy scenario of cost and waste. The hidden danger in public information infrastructure was the subject of an article just two weeks before, as I wrote this, in the *New Scientist* magazine. This is true to life and no one's 'fault' but does need to be recognized for what it is. Unfortunately, too many high-level minds set off to build cardboard bridges across turbulent rivers, employing a workforce of skilled but non-swimming stone masons. They throw their hands up when all seems lost, just a few metres from the bank, and proceed to plan a tunnel, which turns out to be through granite and bankrupts the Treasury with the cost of diamond rock cutters!

In case you doubt this graphically polemic language, investigate the story of the planned replacement, twenty-five years ago, of the UK west coast railway line, described as a '£10 billion rail crash'! Googling 'signalling west coast main line 10 billion disaster' should do the trick. In this plan, a budget escalation, a massive amount of which was hypothecated to expensive wired signaling systems, was 'solved' by a decision to do away with these systems. The project was given the go-ahead by the government, with the condition that it would implement a computerized wireless signaling system (connecting between trains and a central control room), which at that time were non-existent and thus operationally unproven. The project collapsed at that time, the congestion of traffic remained unsolved, and is

now the target of the one hundred-billion-pound HS2 high-speed rail link from London to the North of England.

You might also come across another project, nearer to home, characterized as a '£10 billion NHS IT project disaster'. More on that in Chapters Seven and Eight. Software ecosystems are complex socio-technical edifices; an interface of user requirements, organizations and ways of working, with engineering methods, skills and tools, employing development and maintenance teams, management and money. Planning for their design and implementation shares the features of the wicked problem of social policy that features in my critique of the architecture of health care information systems, in Chapter Eight. This chapter's engineering focus now moves from software methods to systems architecture.

System and Architecture

One can make a car from scratch by breaking the task down into multiple component actions of design and production and setting up multiple departments to enact them: for wheels, engines and parts of engines, chassis, bodywork and internal fittings. One can repeat this for successive car models, each designed to catch the buyers' eyes and entice them with new features.

Someone is needed who has learned and knows about users of cars and the needs and preferences that influence their purchases, and about the car as a whole–how it is built and performs, where and when it will be driven, how safely and how it feels to its occupants. Just as buildings need architects and designers, so, too, do information systems. They bring a higher-level perspective of the human needs they serve. Such architecture is hard enough when making a product like a car or a submarine, but tractable with experience, authority, money and time. For the enterprises of health care–evolving rapidly, alongside human lives being turned upside down, for patients and carers, for their professionals and for the funders, suppliers and regulators involved–the complexity in the making is of a different order.

Architects are chiefs among creators of systems, as archbishops are chiefs among bishops. Architecture is associated with the names of its architects, and architects create and belong to schools and fashions of their times. There are professional architects of buildings and ships. Information architecture is the bridge between the requirements and design of information systems, and the glue bonding together their implementation and operation.

Architects of buildings work from a brief that sets out a vision of the requirements to be met by a new building. They envisage and formulate plans for how that vision might be achieved, within specified constraints of

applicable regulations and available money, materials and workmanship. They talk the languages of users and commissioners of buildings, and of designers and constructers of buildings. They draw on a pedigree of expertise, experience and reputation. There are guidelines that inform their proposals for access, space allocation, insulation, heating and lighting and so on, to support the intended uses. They combine materials and structures to create an aesthetically pleasing functioning environment. At the heart of each work of art there is an artist.

Painters start from a blank canvas and architects of buildings may often be asked to do so, as well. 'What might a family home look like at this clifftop seaside location we have just purchased?', a wealthy family who have just won the national lottery, might ask. Information architecture in support of health care has also developed from blank canvases along my songline. 'Make a computer do this' is a blank canvas to work on, where what 'this' is, and its operational context lack clarity and consistency. How best to represent requirements, plans and designs, such that they can be safely realized by system developers, in the anarchic context of the information revolution and its mushrooming and chaotic technologies, has often been anyone's guess.

It is a situation akin to that described in the age-old tale of the car traveller–that I have already invoked, in the Introduction–who was lost when trying to reach Dublin. They stopped to ask directions from someone standing at the roadside, who thought for a moment and then replied, 'If I were you, I wouldn't start from here!' This story is somewhat near the knuckle, as a metaphor for information architecture in our era. It is the situation facing many an information system architect, asked to show how the goals that have been set, and how they can and will be achieved. The destination, as defined, lies somewhere at the end of a rainbow; the car in which the traveller expects to reach the destination is destined to break down irreparably within five miles, and is, in any case, almost out of fuel; the road ahead is beset by floods and landslides, and some has not yet been built! Information architects are often troubled by not knowing where to start. In my lectures of those times, I used statistics quoted in the December 1996 *Software Magazine* about failed IT projects. It reported that, in 1995, more than one hundred and forty billion US dollars was spent on IT projects in the USA that were either cancelled or 'redefined'. And eighty four percent of projects failed to deliver what the stakeholders needed and wanted. Judging by recent reports, as discussed in the Introduction, this reality has not improved.

My first encounter with information systems was in the heavy engineering industry of the mid-1960s, when systems analysis was the focus and paradigm for representing work and data flow within organizations,

with a view to improving or streamlining them more cost-effectively. Formal project management and control methods split component sub-tasks within integrated production processes, along time critical pathways. In later years, I learned, in similar manner, how the modelling of bodily functions differentiated and atomized them into respiratory, cardiovascular and endocrine subsystems, and many more. In the body, no such differentiation pertains–the system works as one entity, a whole. High positive airway pressure applied in the intensive care unit to promote respiratory gas exchange, may also apply a reverse hydrostatic pressure inhibiting the circulation of venous blood back into the heart and lungs, thus countering delivery of oxygen and removal of carbon dioxide throughout the body. Managing the complexity and interconnection of biological systems led in time to the integrative study of systems biology and systems medicine. Although integrative in intent, these new entities led to new boundaries and contexts of discipline that were rather arbitrarily defined and defended.

In the world of engineering, information systems became systems of systems. Everything was broken down into components and connections, rather like complex electrical circuits. Within organizations, patterns of working practice, mixing actors, actions, products and services, were laid out on extensive flow charts and maps. The goal was towards reorganization and automation of working practices, enabled by computer systems that would carry out what might at one time have involved pen, paper and filing cabinets, with communication via postal and telephone service. Just as data could, and needed to, be modelled in various ways to reflect different purposes and perspectives, the same applies to the related information systems when specifying how the data were to be collected, analyzed and used. Having atomized the tasks to be performed into discrete components and connections, developers proceeded to automate them by writing software that represented and enacted this design. There was no way of knowing how the combined new system of people and computers would work in practice, whether the computational part was reliable and efficient, and the human part comprised satisfactory and sustainable work.

This era exposed limitations arising from the continuous and chaotic evolution of methods for representing and programming computations and information flows. The continuous evolution also extended to computational, data storage and communication technologies. There was frequent frustration with failures to understand the requirements for a system that could and would do the job it was envisaged it would do, at the time it was expected to do it, at a cost that had been budgeted for, staffed and operated by people available and trained to use it. Mismatch with the users' needs, capabilities and expectations led to a culture of default. Expression of user resignation defaulted to the *Little Britain* TV show actor

David Walliams and 'computer says no'! Failures identified in the workings of organizations and society, more generally, were ascribed to problems of 'the system', deeply entrenched and termed 'systemic'.

In the hospital context, one must only suffer the misfortune of lying for an extended period in a hospital ward–or sit, anxiously, with a very sick relative, over long periods of months, as I have done, or attend clinics where little is known that connects with why one is there and what is or should already be known about a patient's presenting condition–to understand that fragmentation of computer systems is mirrored in atomization of services. A close colleague wrote to me, recently, after his own similar experience as a hospital inpatient. Tails of poorly conceived and executed computerization have wagged dogs of attentive clinical care. Extremely hard-pressed professional and clerical staff pore over badly designed screens, battling demands for entry of data that disappears, to appear later in spreadsheets or orders for actions to be taken.

This burdensome practice cannot coexist at all well with an orderly and reassuring environment and atmosphere, at the point of care, in the clinic or on the phone, answering to the anxiety of patients and their relatives, distinguishing between the real urgency of pressed alarm bells and the false alarms from bedside monitors. These are systems of poor utility.

The father of utilitarian philosophy was Jeremy Bentham (1748–1832)–utility as an end guided by and pursued for the good it confers. Common services, such as for energy, water and communications, are utilities; as Joel Birnbaum said, they function best when least noticed. His worldview of information as a utility is discussed in Chapter Seven. Statistics quantifies utility of an action, such as a clinical intervention, in terms of probabilities and values ascribed to its various potential outcomes. We often seem caught, confusingly, between high probabilities of low-value outcomes and low probabilities of high-value outcomes. Confusion and disconnection of ends (what is aimed for) and means (how it is approached) can readily lead to the creation of costly information systems that prove of low utility. How much money was spent on track and trace systems during the Covid pandemic, I wonder? This chaotic and wasteful panorama is long-standing–what are its mitigations and remedies?

Fred Brooks was Professor of Computer Science at the University of North Carolina (UNC) and worked in the mainframe computer era at IBM as architect of the famous IBM/360 series computers. He was a colleague of Edward Feigenbaum's collaborating team at Stanford and is widely known for his book, *The Mythical Man-Month*.[47] In this he distilled his professional

47 F. P. Brooks Jr., *The Mythical Man-Month: Essays on Software Engineering* (New Delhi: Pearson Education, 1995).

experience of an era where engineering of systems became rather lost in the woods. He pinpointed the consequences of poor architecture and implications for design leadership of software systems.

'Systems need architects' was one of his memorable aphorisms. Another was: if you are falling behind in making a new system work, adding numbers to the team can easily make matters worse. In coining the term, 'mythical man-month', he was highlighting that innovation does not come in quanta of months of effort. As experienced with the RSX operating system toothing troubles in DEC, which I lived and worked though, a small team can move faster and crack open and solve problems that have defeated much larger teams, even ten times the size. The large team becomes so consumed in internal communication about the problems and failures, as to be unable to solve them. The stripped-down team, liberated from an incumbency that has grown to tackle and cover for confusion, works together more easily, and works things out.

Man-months do pay salaries, however, and there is perverse incentive in play. New hype and mantra expressed in the pursuit for new resource wins often befuddled political support. Politics commands public purse strings and seeks to show how modern and imaginative it is. Innovative and disruptive thinking does not often win in this arena and is seen as an opponent to be suppressed.

Failed engineering projects often also result from a combination of invalid assumptions and undue expectations. These lead to poor decisions about method to be employed, capability and planning. And the diagnosis and blame for failure is often attributed to somewhere within the technical domain, where indeed the symptoms often do arise, with cost and time overruns, key expectations not met and systems performing poorly. In large-scale information systems, failure is evidenced in wide-ranging breakdown and malfunction, akin to sepsis or proliferating bodily cancer. Its causes may rather reside not in a localized defect but in the architecture of the whole system: of knowledge, actions, roles and responsibilities. From ambulance to recovery ward, data on a single arriving patient may pass through tens of separate computer systems. It does little to improve continuity and coherence of care, and services of care. An atomized set of systems adds administrative cost and workload along the line.

Uncovering the detail of historic software systems is like visiting an archaic domain–akin to the library in Umberto Eco's novel, *The Name of the Rose*, with disconnected custodians preserving its hidden away and guarded secrets.[48] This is software entropy. Antiquated software often reveals itself

48 U. Eco, *The Name of the Rose* (London: Pan Books, 1984).

like a rusty old car tottering along the road and coming to a teetering halt. Along the road are many abandoned burnt out wrecks, blocking the way ahead for newcomers.

Delving beneath the bonnet of an archaic information systems, one may find a morass of rusty databases and smoking information circuitry. It may not catch on fire, but it has grown old and dysfunctional through accumulation of entropy. The code is populated with string and sealing wax software patches that have been hurriedly added, to meet new needs and cover previous defects, and then forgotten about. It creaks and judders, having grown old and out of tune with changing methods in the vanguard of the industry. Whole industries have existed to keep such legacy on life support. The original coder may have long since departed, leaving little if any documentary record of their programs–not just what it is but why it is as it is and the compromises and limitations it embodies. Sometimes, the code can no longer be compiled from source to machine-code form, and exists only as an impregnable binary object, and encryption which may defy even costly reverse engineering.

Many of us have experienced such information system jalopies and car crashes. I have been thanked for writing this, here, as it was. It is not something to be conveniently forgotten! Fifteen years ago, I was asked to chair an NHS group set up to monitor implementation of a plan for common prescribing practice across all general practice (GP) and pharmacy IT systems. It proceeded at a snail's pace as each supplier struggled to adapt their code and databases in line with the new common requirements that had been nationally mandated. It was an archaic and painful process, far removed from the promise of agile development that was the mantra of the age.

Another illuminating example of the general weakness of system architecture came with the looming arrival of the year 2000. This had engendered a peak of concern about the robustness of software and demand for perusal of code, to search for and iron out weaknesses arising due to the way all dates were being recorded and processed in programs and databases. A common programming heuristic of those times had been to represent and store the year as two digits, implying a date relative to 1 January 1900–or 1901 if 00 was reserved for 'date not known'! In this model, as the clock struck midnight on 31 December 1999, time would come to an end and the recorded date of a new entry would be reset to 'unknown' or 0. This threat was perceived as a pending national emergency and a huge amount of time and resources was spent scanning through virtual acres of program listings, throughout the economy, to find and fix date representation problems that might be revealed there. There was bated breath in high places as the striking Big Ben clock–which the dangerously

crushed crowds on the Thames riverbank were waiting for, expectantly and drunkenly–hailed a new century. The wave of fireworks propagating along the river, that had been promised to celebrate that same moment, proved a damp squib, and the feared systemic breakdown of software throughout the economy did not happen either. This confirmed the prediction of my colleague, then UCL Professor of Computer Science, Anthony Finkelstein, who had risked reputation and demonstrated his clear head and safe pair of hands, in sagely opining that it would be that way. He went on to become a government scientific advisor and then University Vice-Chancellor at the alma mater of his formidably learned and accomplished father, Ludwig Finkelstein (1929–2011), a doyen of the field of measurement and control engineering who, with his protégé, Ewart Carson, had a special interest in medicine.

The key learning from this story is not that the feared Year 2000 disaster did not happen, but rather that there had been so little confidence in the industry and user domains, that information systems would prove robust to this obvious challenge. And yet, some good engineers and leaders of engineering were certain, without investigation, that their own systems would not prove vulnerable. One such was my colleague, Jo Milan, architect and designer of the nationally preeminent cancer information systems he and his team built and sustained throughout most of my songline, at the Royal Marsden Hospital, in London and Sutton. Why and how could he be so sure? Because, as Jo explained to me, the coherent, concise and superbly functional data models that underpinned the whole of the information systems of the Marsden (my words, not his, he was a very modest man; I tell his story in Chapter Eight), invoked just one very small date function. He had written it and knew it was robust. To satisfy his untrusting hospital managers, who failed to recognise the jewel in their crown that Jo and his team and their systems represented, Jo did pull it briefly to his office screen and took a few moments to scan and verify its correctness, before confirming in an email that this had been done–keeping his managers in the clear, upwards in their hierarchy of NHS managers.

Teams responsible for supporting systems with much less complex requirements, but exemplifying code threads more akin to a pile of spaghetti, dedicated many months to this task, diverting their attention from other important work. There are immense burdens of cost and opportunity imposed everywhere by poor information engineering. Many projects never make the transition into everyday practice–the first Covid-19 App of NHSX is one from today. National bank software systems have crashed and malfunctioned several times over recent years, their maintenance and update procedures unsound, bedevilling both businesses and public services.

And information system security flaws expose their user organizations to unavoidable ransom demands. Knowledge of flaws in software and databases does bring opportunity for the virtuous, as well as for the less virtuous, on occasion. I remember giving a talk at a Research Council strategy board meeting, and listening to another speaker there, from an eminent bioinformatics research group. The discussion was about errors in genome sequence databases. In a seemingly not entirely tongue-in-cheek manner, they remarked that teams sometimes did not quickly report errors they discovered in shared databases. They could adjust for these for themselves, but competing scientists might not be aware and would be left to independently discover them, at a cost to the accuracy of their own analyses and the productivity of their work!

Science is well equipped with intelligent minds, well able to look after themselves and the impact of such error and imperfection within science is unlikely to cause a Challenger Space Shuttle scale of disaster. But health care services are more vulnerable. They exist to provide a human service, keeping on top of scientific advance and increasing and changing demand, and must contend with incoherence and complexity of information. Overloaded senior managers easily lose sight of systemic problems and of the human efforts and values that hold everyone's efforts together within the service. As pressures escalate, it is understandable that highly motivated people lose heart.

Health care policy makers have sought to protect the service risks they are accountable for, by substantially outsourcing problems and their solutions to others. It is a bit like outsourcing any service central to the running of a business. Common good advice is only to outsource things you understand and that are not core to what makes you special. If you do, you are admitting to a weakness and setting yourself and your business up for mistakes and exploitation. And no amount of court cases and governance checks can put right the damage that may ensue. This adds further cost and intractable complexity. Health care services have endured and accumulated both.

Solutions to wicked problems can only be found in cooperation and collaboration–from the centre and from the ground–in the clinical and caring professions and with the patient communities they serve. This will be a trajectory of David Goodhart's head, hand and heart, to generate new roles and capabilities that will be needed on the other side of the Whitehead–and in the UK context, Whitehall–information systems anarchy of the past fifty years.[49] That is the trajectory that I trace and anticipate in Parts Two and Three of this book.

49 D. Goodhart, *Head Hand Heart: The Struggle for Dignity and Status in the 21st Century* (London: Penguin Books, 2020).

Norms, Standards and Standardization

> The essential thing in form is to be free in whatever form is used.[50]

On holiday in the small seaside town of Port Bou, in Northern Spain, the flat where our family was staying in the 1980s overlooked the beautiful beach and sea, on one side, and the railway line emerging from a tunnel from Perpignan in France into Spain, on the other. The intercity express trains crept very slowly into a long shed covering the tracks. Emerging at the far end, they accelerated away. On the inside of the shed, the wheels and axles of the bogies suspending the train carriages underwent a conversion between two national standards. To the north, the gauge of the railway track was that of France and to the south that of Spain. The diverging/converging rail tracks going each way through the shed, combining with the bogies, performed a standards conversion and the wheels emerged with new separation along the axles to conform with the new standard required. A bit complicated and time consuming, and maybe frustrating for passengers eager to get to Barcelona, but better than a train wreck, coming off the rails at the border between countries! A quick check on Wikipedia indicates this divergence of standards still pertains, forty years on. It shows how difficult it is to shift and modernize infrastructure. Information infrastructure is no different.[51]

I read an equally illuminating story, recounted by the historian, Norman Davies, about Russian history, and the merger of armies after a war. Weapons and ammunition from the opposing armies were collected and pooled in preparation for new campaigns, only to discover that rifles and bullets did not align; the precision necessary in this case is likely much greater than that for train wheels. This was one of many complicating logistical and organizational problems arising from the conquest and the merger of armies. Another case of standards not aligning, with potentially explosive consequences.

---

50 Wallace Stevens (1937), quoted in C. J. Date, *An Introduction to Database Systems* (Delhi: Pearson Education India, 1975), p. 263.

51 I came across another connection with Port Bou when listening to a recent podcast about Walter Benjamin (1892–1940), a highly influential German-Jewish literary critic and sociologist of culture of the past century. His philosophy was a practical one, seeking to connect his thinking with everyday life and experience. He was not a fan of wordy and generalized abstract thinking, saying that 'I have nothing to say, only things to show'. Clearly an engineering minded philosopher, and a committed networker. In 1940, he was escaping from Paris and the German army's advance and proceeded to the Spanish border at Port Bou. He was refused entry, and committed suicide there.

The terms 'norm' and 'standard' are loaded with meaning. *Norma* in Latin meant square, and we still talk of square meals and square deals, implying something appropriate, balanced and fair. As discussed in relation to the theme of measurement in Chapter Three, according to Ivan Illich (1926–2002), nineteenth-century English geometry took over the term and normal came to mean to be at right angles. A connotation of principal axes spread into normalization and normal forms and by the late nineteenth century had come to symbolize conformity to a common type. These terms have acquired specific meanings and appropriations within the language of computer science and database design. In France, the École normale supérieure was established to train teachers in correct usage of the French language in French-speaking countries. Likewise in France, Auguste Comte (1798–1857) introduced a new medical connotation to the term, and, by the end of the nineteenth century, these norms became bound up with criteria for diagnosis and treatment of disease.

The term 'standard' also had a more proprietorial tone. Something to be expected in polite society and in the husbandry of animals and crops, and of resources, more widely. They became components of more formal governance; rules that should be adhered to in pursuit of social and material goals, allowing life to function harmoniously, and things to fit and work together well. With a motor-driven electric power generator situated in the garden, supplying the house, it did not much matter what exact voltage it supplied to power domestic appliances, provided one had a kettle or lights that were compatible. Providing electrical power as a utility for the whole village and nationally, it again did not matter a lot; there were pros and cons of different transmission line technologies, but what did matter was that there should be one standard that providers and consumers adopted and adapted to.

Technical standards like this serve many purposes in design, development, manufacture and supply of goods and services, and in simplifying their efficient updating and maintenance; this contributes to the creation of a coherent workforce of support engineers, trained and up-to-date with the technology. Such standards can simplify work, enabling the realization of economies of scale and providing a seamless service, where disharmony of incompatible products and services imposes unnecessary costs and increased overhead.

Standards for information systems enter wider realms of complexity and contention, reflecting their wide and pervasive contexts and the methods adopted for creating, making, using, maintaining and regulating them. There are good and almost unanswerable points in favour of standardization of some areas of endeavour. It is difficult to see why any developer writing software for an information system embodying representation of time

would think it a good idea to reinvent ISO 8601, the international standard covering the exchange of date- and time-related data first published in 1988. Perhaps physicists probing towards Planck limits of time might think it largely irrelevant to their needs! However, the choice and adoption of information standards, more generally, has proved a costly, time-consuming and difficult area, exposing and brokering among many differences of perspective and conflicts of interest. I have seen it first-hand, as a wicked problem of health care IT, and will return to this topic in Chapter Seven.

Often in the domain of standards-making, the process adopted to arrive at a consensus about the standard reveals more about the purposes it will serve and ways of brokering conflict of interest about these, than it does about performance of the standard when implemented. Standards have spread from properties of devices–where the scope of a proposed standard is more easily defined and policed within a predominantly scientific or technological domain–to properties of systems and services–where scope and purpose are more widely open to debate and disagreement, on other levels of commerce, policy and law.

Before embarking on the challenge of defining a standard, it is well to establish a basis for discussion about what is to be standardized, why and how. It is not a good idea to start a process towards standardization from a blank canvas of the field to be standardized. In discussions of information systems, there is a need to focus on what is often called a reference model. According to the Organization for the Advancement of Structured Information Standards (OASIS):

> [a reference model is] an abstract framework for understanding significant relationships among the entities of some environment, and for the development of consistent standards or specifications supporting that environment. A reference model is based on a small number of unifying concepts and may be used as a basis for education and explaining standards to a non-specialist. A reference model is not directly tied to any standards, technologies, or other concrete implementation details, but it does seek to provide a common semantics that can be used unambiguously across and between different implementations.[52]

A reference model provides information about a particular kind of environment and the types of things that exist there, how they mutually connect and interact with one another. It is an abstract model and does not talk in terms of specific methods of implementing them. It is not in itself a standard but can provide a framework for standardization, creating

52 'OASIS SOA Reference Model (SOA-RM) TC', *OASIS Open*, https://www.oasis-open.org/committees/soa-rm/faq.php

standards that ease the work of developers and making them more useful and applicable in wider contexts. In that a standard provides a basis for discussion, good standards can play useful roles in education, communication and organization. When different approaches to standardization are under review, it is good to have concrete and implemented proposals at the centre of debate, where they can be compared, when reaching decisions.

Standards-making processes can all too easily become contentious and bogged down, papering over differences and forcing resolution based on votes cast by representatives of different interest groups rather than on a basis of both theory and implementation practicality. In the political domain, they can be used as instruments of control and manipulation, promoting interests of proposers of a standard, and blocking those of their competitors. They may appeal as convenient garments to clothe emperors, who use them in name only, as cover for their lack of knowledge and experience about the basis or impact of the standard in practice.

Viewed top-down, there is a strong temptation to tackle lack of practical understanding of a domain by prior assertion of an answer about how it should be standardized. Better by far to pursue standardization through incremental and experimental method, leading to the definition and adoption of specific standards, based on both evidence and declared intent. That is the basis whereby science can persuasively align its theories and engineering usefully its designs. No process of standardization can disguise or compensate for lack of understanding of a domain, but confession of that reality is not always seen as an option. The pretence of knowledge is maintained through a combination of abstract confabulation and wishful thinking. Benjamin Disraeli (1804-81) once described a windbag colleague politician as being 'intoxicated with the exuberance of his own verbosity'.[53] In the Information Age, echoing Mervyn King's remarks about damaging hubris and pretence of knowledge in the world of high finance, practitioners of the Information Age are too often overloaded with the burden of inconsequential detail.

We rely on a supply of electricity, gas and water, to flow in networks and arrive at a standardized interface with house and home. We need information that flows, just as water flows to create and support life. The electrical engineering of information flow is now a well-designed, polished and maintained infrastructure, although with continuing scope for improvement. The flow of information, with its causative power, can be a turbulent flow of rivers, bursting banks and flooding across landscapes. This can be a destructive power, different in kind and feel from that

53 Quoted in *The Times* (29 July 1878).

appealing to the optimism of David Deutsch, about the power of knowledge (as information with causative power) to resolve the ills of the world. In calmer water analogy, the experience of the Alhambra Palace in Granada, on a hot, sunny summer day, is overwhelming. Here is beautiful architecture and design, here is free flowing and calming water. Here is a simple and functional engineering system, delivering water from the mountains behind to the fountains, in the sunlight and shade.

Information flow is not sweetness and light; it will always exhibit bias and inaccuracy, and harbour destructive and criminal potential. It will be contentious and there will be competition. But it can be a lot better than at present. It has become increasingly corralled, entrained and attacked within and between adventitious global monopolies that then seek, first and foremost, to protect and preserve their own expanding ambitions and interests. Standardization of information for health care needs to grow and embody a better balance of global and local perspectives; the local anchored foursquare within local community and democratic politics. That square should be the new norm. There is much work to be done. It is to that square that the book heads in Part Three.

Information systems nowadays are procured and bolted together from products of industries that address international markets. The purchasers' choices are binary: between a contract to do it all and a suite of contracts to piece it together. These require different skills sets, resources and appetites of those making decisions on purchases. The former requires very deep pockets and willingness to tolerate the supplier's system largely as is. There exists only limited, and expensively disincentivized capacity to adapt locally. The latter approach requires methods and local capability and capacity to support local standardization, at a level that is not yet easily achieved.

When researching the purchase of a new car recently, I read that the full range of Volkswagen cars now rests on a production platform of modular components that fit together in their product range of cars. The benefits extend throughout the business: in design, production and maintenance, and in the back-office as well. In developing the theme of this book, I am seeking to show how we can reach this win-win in health care IT. In this, we need to marry the benefits and imperatives of global coherence, with the needs and interests of the local, marrying together both the local and global villages of health care we now populate. For this, we need good standards that promote what has been described as 'co-opetition'–the combination of cooperation and competition.

These two villages form the starting point for my discussion of the evolution of health care in the Information Age, in Chapters Six and Seven. I will describe examples of coherent information architecture, pulled together across major hospital systems and the benefits and user satisfaction that

these have created. In Chapter Eight, I devote a half chapter to two open initiatives I have nurtured and supported from their beginnings, a prelude to discussion of Open Data and other 'open' movements in Chapter Nine. The first is the story of the openEHR mission to provide open specifications for a generic platform architecture for health care records, now lifting off in implementations around the world. The second, which I introduce more briefly, is the OpenEyes open-source software application. Adoption of the current release is accelerating and already providing records for approaching fifty percent of eye consultations in the UK, including in national programmes for the whole of Scotland and Wales.

In Chapter Nine, I come to how we can frame and lead the combined pursuit of continuity and change in the Information Age, from Whitehead transitional anarchy through to a new local- and global-village order of care information utility, in a manner that promotes what Robert Axelrod described in his book, *The Evolution of Cooperation*.[54] This may be destined to come about chaotically and destructively, driven from science or from society. It can, though, come in more stable and just ways, at their interface, supported through the rigorous, engaged, and trusted efforts of engineers and engineering. Which path plays out will depend on how the challenge is approached–whether inclusive or exclusive. My anthropologist colleague at UCL, Paul Bate, surveyed implementation of health care innovations. He described what he saw as an 'implementation gap' in health care innovations and quoted Donald Hambrick and Albert Cannella,[55] who observed 'outcomes and levels of success that are less–sometimes considerably less–than originally planned or predicted'. He concluded that 'it is not so much the improvement method itself, or even the strategy, but how it is done (and in the case of imports, how it is customized) that determines ultimate success'. The music hall song line 'It ain't what you do, it's the way that you do it' comes to mind! Especially so, in the context of wicked problems!

The Internet and World Wide Web

The arrival of the Internet and World Wide Web have had a powerfully formative influence on almost all areas of information engineering and standards impacting on information systems. Volumes of data stored in Cloud data stores and accessible throughout the world have multiplied from petabytes to many multiples of exabytes. Iceland became an early home

54 R. M. Axelrod, *The Evolution of Cooperation* (London: Penguin Books, 1990).

55 D. C. Hambrick and A. A. Cannella Jr., 'Strategy Implementation as Substance and Selling', *Academy of Management Perspectives*, 3.4 (1989), 278–85.

for ice-cooled data stores, and the Microsoft Azure cloud is implementing new datastores, cooled deep undersea. Data storage with Azure is held in triplicate and the network of linked processors offers computational resource on virtual machines, worldwide. Google Docs, Amazon Web Services and Apple iCloud have similar Cloud infrastructures.

The explosion in number and scale of data sources and the standardization of browser technology interfacing with users of systems, has brought new paradigms of data management, programming language and client-server architecture of applications. The scale and diversity of large datasets captured in scientific research, has tested data management paradigms to the limit. The scale and complexity of processing required in analyzing data has forged new priorities for networks of computers and programming methods that allow computational tasks to be shared among them. There is a growing preponderance and focus on large datasets, hosting what is termed unstructured data, annotated using standardized markup languages, but non-relational in form.

This revolution has placed pressure on the working practices of standards organizations. Internet timescales could not survive the five or more years in which an International Standards Organization (ISO) standard typically took to mature and reach publication. The demand for immediate results, in the context of both software and standard, became both imperative and declarative: 'I don't want anything in particular, but I want it now'!

ISO has deprecated detail derived from specific implementation, wishing to remain vendor- and product-neutral in its standards. Its constituency has been based on 'one country, one vote', and is thus, inevitably, a political mix. The Object Management Group (OMG) was born out of industry frustration with the ISO process, as an industry collaboration to meet industry needs, and voted on by its member companies. OMG happily takes on board methods of standardization where there is an existing implementation, if these can be freely published and accessed by all its members, who pay a fee for their membership to cover the costs of the organization. To cover its costs, ISO makes a charge for downloading of its standards documents. In many cases, OMG standards have progressed, verbatim, as fast-tracked ISO standards, but some years later.

The missions and goals of OMG and ISO are quite similar in their wish to facilitate standardization to the benefit of all sectors of the economy. But their methods, constituencies and loyalties differ considerably. In the realm of politics, determining international standards from on high, the ISO method lines up with national preoccupations and concerns. In the realm of industry, the pursuit of business opportunity favours the OMG approach. OMG is investing in the concept of architecture driven modernization of systems and services, providing tools based on a Business Architecture

Core Metamodel (BACM), to align and support suppliers in modernizing their products within the changing information landscape.

The address hierarchy of Internet-connected devices and services has been under the management of the Internet Corporation for Assigned Names and Numbers (ICANN), a not-for-profit organization responsible for Internet Protocol (IP) and Domain Name Services (DNS) that transport and navigate data throughout the Internet. Originally established in different form under US Government contract, the organization works collaboratively across all countries and sectors, to maintain the global Internet.

An IP (Internet protocol) address is a numerical code divided into sections, which enables electronic connection to the domain of a specific device and location. To avoid or minimize unnecessary use of complicated numbers, this allocation is managed by the DNS as a symbolic domain name, mapping between symbolic names of domains and their IP numerical sequences. If I am a domain, my house number, road, city and country might constitute my IP address. If I move to a new house, I can take me with me, but my current IP address stays where it is. I can reroute my postal mail by notifying the postal service of a new location. My Internet communications are routed to the IP address of any location where I log on.

Skipping back to information systems, resources hosted within devices and domains need unique identifiers, called Uniform Resource Identifiers (URIs). They are strings of characters that unambiguously identify a particular resource. They follow defined syntax and are extensible to unique identification of separate resources within subdivided domains. A Uniform Resource Locator (URL), or web address, references a resource by specifying its location on a computer network and a mechanism for accessing it. URLs are used most commonly to reference web pages (http: and https:), but also for file transfer (ftp), email (mailto), database access (JDBC) and many other applications.

The Status Quo

In the 1960s, pioneering NHS colleagues that I first encountered–like John Anderson (1921–2002), a professor of medicine; Frederick Flynn (1924–2011), a head of chemical pathology laboratory services; and John Clifton (1930–2023), my chief of medical physics at UCH–needed all their skills of management and persuasion to deal with a health service that was then placing all things computer in its central Supplies Division, with staff overseeing purchase of chairs one week and computers the next. How health care has navigated through now seven decades of scientific and technological advance, in and through the Information Age and its

associated reorganizations of services and strategies, is an extraordinary story. The convergence of politics, commercial interest, hubris and pretence of knowledge, ambition, immature new technologies and practical realities of health care can create chaotic situations akin to Macbethian cauldrons of 'double, double toil and trouble'! Burn, bubble, boil and bake sounds about right, not to mention sting and charm! Where connection and influence are global, what might once have been contained and localized, becomes of global impact. A detailed account of the coevolution of health care and information technology is rehearsed in Chapter Seven.

This final section of the chapter takes a brief look at how evolution in information engineering has impacted the governance of public services, with a story of a seemingly common misadventure uncovered in one government ministry. In company with many large organizations, governments have come to realize that they have not been good at designing and implementing information systems. Of course, such difficulties are not unique to health care.

In the late 1990s and early 2000s, I worked closely with Al Aynsley-Green when he was director of clinical research at the Great Ormond Street Hospital for Children and UCL Medical School. He led the creation of a database of all the research teams there, and their projects, to assist in the development of the research plans at the Institute of Child Health.[56]

Al became a supportive colleague in this project. He was subsequently appointed as the first Children's Commissioner for England, in which role he took an interest in information systems at the government Department for Education (DfE). One initiative at the time had seen external consultants brought in to analyze and propose improvements in its information management. Inevitably, this extended into schools and universities, courts and hospitals, and interfaced throughout government departments. Uncovering rock by rock, as on a seashore, to see what lay beneath, the

56 The then chief of medicine of the UCL Medical School, Leon Fine, asked me to help in creating something similar for all of the UCL Biomedicine Division (which was, by then, accounting for about a half of the one billion pounds annual financial turnover of UCL), and its linked NHS Hospital Trusts (which multiplied this one billion several times over). I could not say no, of course! It was a tough ask, but a good way to get to know the wider institution and community. I was looked to, to take on several such aspirational tasks and integrative roles in those years. Though distracting from a more focused academic mission, these roles were implicit in my appointment and the expectation of those appointing me regarding my broader contributions to the University and local health services. Striking a delicate balance, we were able to create and sustain a rich interdisciplinary and multiprofessional environment for the CHIME department. Its challenges and achievements are reflected on in Chapter Nine, in the context of our experience in creating a new working environment for the Information Age.

consultants discovered an extraordinary complex of databases. Not dissimilar to what, I later understood, was embodied in the large collection of databases maintained under the aegis of the NHS Information Centre.

Al asked me to come to a meeting at the DfE, where the consultants' report was to be discussed. There were serious mandarins in attendance, baffled and bemused by the spider's web of data relationships that had been identified and charted in the consultants' systems analysis. A heap of spaghetti would have writhed in embarrassment! I had come across many such intractable data networks over the course of my career and was sympathetic, asking, simply and innocuously, what they felt they had discovered in the project. The enigmatic response from the department's commissioner of the report was that they now had a 'clearer view' of the chaos! Were this a discovery about plans for an aeroplane that would likely not fly, or a submarine that would likely sink, there would be the option, and perhaps imperative, to start afresh. But no one there felt responsible for the situation in question. All present experienced the reality but did not, or could not, diagnose and connect the problems revealed with how to set about solving them. I imagine they could have done no more than put the report aside and move on to other pressing concerns.

The consultant's very detailed systems analysis had revealed a lack of architecture and design in the many preceding systems analyses that had led to this set of atomized and uncommunicating databases. The use of a consultant to conduct what looked to have been a rather fruitless exercise was a classic example of what Mariana Mazzucato and Rosie Collington have described in their recent book–the final inukbook that I have drawn on in writing this one.[57] It describes the present-day double bind that pushes organizations to commission external consultants to study and diagnose their problems and propose solutions, which end up costing a lot and adding little to the inhouse capability that would be implicit in the capacity to enact necessary remedial action and change.

Such anarchic assembly of databases is widespread. Some organizations have sufficient resources, opportunities and capabilities to wipe the slate clean and start afresh, but most in the public sector do not. In health care, this situation is also demonstrated in the several hundreds of non-communicating, small-scale and isolated systems in use in major university

57 M. Mazzucato and R. Collington, *The Big Con: How the Consulting Industry Weakens Our Businesses, Infantilizes Our Governments and Warps Our Economies* (London: Allen Lane, 2023). The 'Con' in their cross wires is Consultancy, playing with the implication that it can be akin to a confidence trick. Being a bit provocative, myself—to match the outspokenness of these authors in addressing their theme—is there sometimes another kind of contemporary 'Con' in play, captured in their 'Big Con' title—namely, the 'Con' of the 'Big'!?

hospital Trusts. These may, indeed, each be performing essential everyday work. But few have specifications extant, showing how they were designed and implemented. And the software tools used and the computers and peripheral devices, themselves, may mostly now be obsolete. The situation was also revealed in project dissertations of students I supervised, who enrolled on postgraduate courses in health informatics with us at UCL, from their day jobs in hospital Trusts. Projects all over the world have battled with, and sought to integrate, non-coherent software systems. Such an endeavour costs a lot of money and often fails. And even a successful short-term outcome may quickly be rendered technologically or functionally obsolete.

As in these sorts of example, many IT projects, large and small, have defied the wisdom of Fred Brooks and paid too little attention to integrative, health-economy-wide architecture and design of information systems. This has occasioned a very great loss of money, impairing and destroying existing in-house capabilities, while distracting from other health care priorities. Each new such venture can impose an additional burden at the coalface of care and constitute a further burden for its central management, in coping with and unravelling the further complexities and inconsistencies that lack of coherent data generates.

Knowledge and experience that bring the capability and capacity to make and do things in-house, has too often been lacking. Battle-hardened folk, like Mervyn King, author books which are more open about the limitations and failures they have observed and worked through. They provide and encourage a culture of honesty and humility, which is essential in learning to make and do better. This is not about prescience; it is about coping better and adopting more realistic ways of working. Policy for health care practices and services cannot be wholly evidence-based. Its implementation is not a controlled or controllable experiment. It is the navigation of an unfolding future of problems that must be coped with and resolved. In this we need traction, balance, purpose and capability to make and do–the parenthetical topics I have highlighted in my chapter-by-chapter reflections, to this point in the book.

One response to these difficulties, seeking to tame proliferation of new and speculative initiatives, is to require prior evidence of effectiveness. This has the sound motivation to achieve safety and cost effectiveness of health care services, as well as absolve from blame when things go wrong. It has created some important and influential new voices–the National Institute for Health and Care Excellence (NICE) in the UK, concerned with cost-benefit evaluation of new treatments, has been a notable success story.

But undue focus on evidence in a domain that can and must be navigated, while still substantially unexplored, is a bit hard on brave explorers. David Sackett (1934–2015), the father of evidence-based medicine (EBM), who

I first encountered at McMaster University before his invention of the field had crystallized, was clinically and epidemiologically grounded and exploratory in his approach. In the heat of the Covid debate, the evidence adduced, and the predictions made in advising on policy, brought many voices and their conflicting views into play. In reality, no one could have known how potentially black swan events–such as harmful viral mutation, ineffective vaccine development and poor behavioural compliance of the public with quarantine and distancing injunctions–would land. Evidence must be weighed, and nowadays, so must evidence about evidence!

The interaction of information engineering with health care has sometimes felt akin to a goldrush in a warzone. We need to become better grounded in the context of health care services as experienced, and not just as described and predicted to be, by people far from the biting of bullets at the coalface of care–patients, relatives and their supporting professionals. We need to find common ground and stand up for what matters to them, and why. We need to co-create new environments in which information can evolve and be sustained as a utility and not as a technology. We need local people, teams and alliances to carry a global flag. If I know my nine grandchildren well, I trust that their generation will prove to be one that can and will make and do this. They look to be up to, and up for, the challenge of finding the common ground required, and helping to create the future there.

Parenthesis–Making and Doing Things Differently

This chapter parenthesis completes Part One of the book. It is a reflective handover into Part Two–perhaps a Mervyn King-style, audaciously pessimistic perspective, but not without some Barack Obama-style, audacious hope, I hope! Erwin Schrödinger (1887–1961) cautioned that what he wrote risked foolishness, when prefacing his attempt to answer the question 'What is life?', as I recalled in the Preface. 'What is health care?' is equally puzzling and one inevitably risks foolishness when seeking to respond adventurously to that important question, as well. Part Two of the book places these two questions side by side, in context of the anarchy of transition of life science and health care services through the past seventy-five years of the Information Age. Where we have got to with these questions and concerns is central to how we can and must now set about reimagining, inventing and creating health care for the Information Society of the future. There is a lot to reflect on!

Information engineering has become integral to the delivery of health care services and will remain hugely consequential for its future reinvention

and reform. It has involved massive investments and implementation endeavours, in multiple dimensions and directions, globally, played out along an evolving timeline over seven decades. It has been at the heart of advances in health care services and instrumental in bringing them to their knees. I reflect, here, on key issues necessitating the reinvention of services, their interrelationship with new requirements for information systems, and challenges in the implementation of such systems. It is safe to say that success in the reinvention of affordable, safe, effective and sustainable health care services (and their associated disciplines, professions and governance) will depend on success in the implementation of a wholly new concept of care information ecosystem. Implementing and sustaining this, incrementally and iteratively, will be how we learn how to do it. It will not be easy and will take a long time, but it is an essential goal.

It is in the engineering domain, where implementation realities are faced and pushes come to shoves, that things oftentimes go badly and expensively wrong, as we develop and deploy new technology and implement it on new terrain. In evidence of this today, we can follow the many years of delay and budget overruns in the construction of the Crossrail line in London. Or the design failures and subsequent crashes of the Boeing 737 MAX aircraft in the 2010s, mirroring those of the de Havilland Comet in the 1950s, which exposed vulnerability to unstable aerodynamic balance and control, and metal fatigue in flight.

Great expectations are vested in new technologies. They were so in the Comet, as expressed by a government minister at the time:

> During the next few years, the UK has an opportunity, which may not recur, of developing aircraft manufacture as one of our main export industries. On whether we grasp this opportunity and so establish firmly an industry of the utmost strategic and economic importance, our future as a great nation may depend.[58]

In retrospection of what happened with the UK avionics industry, there was more rhetorical clutching at straws than grasping of opportunities in this encomium. People talk today about grasping opportunity in health care in a somewhat like manner. Health care information policy has clutched at straws. We may not always need the engineering that makes modern aircraft, but we will always need that which supports and sustains our

58 Duncan Sandys (1908–87), Minister of Supply, 1952, quoted in P. J. Lyth, 'American Aerospace Dominance and the British Challenge in Jet Engines', in *Tackling Transport*: Volume 3, ed. by H. Trischler and S. Zeilinger (London: NMSI), pp. 81–98 (p. 90).

health care services and their information ecosystem. Safety of health care traverses many more dimensions of complexity than does safety of aircraft.

New engineering methods are intrinsically experimental and developmental, and thus prone to mistakes and reappraisal, as they evolve. Where policy and governance are tuned to, and retransmit, signals from a disappearing era of technology, services and society, they can prove vulnerable and maladroit when responding to, seeking to cope with and adapting to radical change, as several of the examples in this chapter have illustrated. Better framing and implementation of plans is about a culture of realism, agility, learning by doing and ability to learn from mistakes. Significant and costly failures can arise when these qualities are lacking or experiment happens at an inappropriate scale–running before walking, as it were. Of course, some experiments can only be conducted at scale, but they remain experiments, nonetheless. And some problems only reveal themselves when explored for real, at scale, or when they emerge from left field. As the French physician and anthropologist Paul Broca (1824–80) reputedly said, 'The least questioned assumptions are often the most questionable'.

There are major and interlinked challenges beckoning as we progress towards the reinvention and reform of the NHS and health care in the UK, today. In a nutshell, these encompass:

- Demographics–an ageing population and associated preponderance of support and cost in providing care for those with chronic conditions, for which there is no cure;
- Social inequalities–with multifactorial causes and impacts on health;
- Separation of health and social care services–despite sharing in common many citizens that they both care for;
- Ineffectiveness of interventions and services–much that is done is deemed to achieve poor cost-benefits;
- Discontinuity–fragmentation of specialisms and professions, leading to disconnected and redundantly repetitive processes;
- Overburdening of workforce–a crisis of ends and means and expectations at all levels–with much human need ending up inappropriately positioned as health care service workload;
- Affordability–an ever-increasing range of improved but costly therapeutic options;

- Rapid and far-reaching changes in life and medical science and device technology–these create difficulties in keeping abreast and up-to-date;
- Burdensome, mutually non-coherent and inflexible information systems;
- Physical estate–much is decrepit and not fit for purpose.

Looking forward, policy is increasingly focussing on:

- Awareness of experience of care and what matters to citizens, in context of quality of life as well as cure of disease;
- Prevention, early intervention and self-care;
- Integration of services at a local level;
- Delegation of authority within health care teams and professions, to prescribe and enact interventions.

These policy perspectives reflect core information engineering challenges:

- Care information systems must be reinvented to be centred on the citizen at home, in the context of care services that connect with them there and within their local community, enabling and eliciting their participation and feedback;
- Data sources are currently highly fragmented and noncoherent, discontinuous, serially redundant and centred on services not citizens. This issue must be addressed head on, as a public domain concern and with suitable new governance;
- A new concept of citizen-centred data repository and related computable knowledge resources must be developed from the ground, gradually supplanting the current legacy with, wherever possible, globally shared and governed public domain specified methodology. This endeavour will target improvement in the cost-effectiveness of information services that directly support health care. It will enable an iteratively and incrementally evolving and improving, and sustainable, marketplace for products and services. It will provide coherent context of data supporting health care governance, management, professional education and research;
- Data volumes have exploded in size and databases have grown serendipitously. The workload of capturing data in the context of everyday health care delivery must be massively streamlined, to

> enable more time for engaging with the need for more personal empathy and care.

There are several principal and related challenges embodied in all this.

First, the current mismatch of need and capacity is not sustainable with current concepts and models of health care services. Social care cannot continue to be envisioned and enacted independently of the health service. There must be a better way and creating it must be a central goal that focuses efforts on the support and enablement of citizens and carers, to participate in and manage the meeting of their needs, at or as near as possible to where they live. Closer bonding of health and social care services requires new professionalism, environment, teamwork and community, and new information systems that mirror these needs. Health care services are basic to society and must have their foundations in teamwork that is anchored and supported more strongly at local level.

Second, the principal information engineering challenge–which will be a long-term, iterative and incremental one, implemented in a spirit of learning by doing–is to create and sustain a locally customized and globally standardized care information utility. This must be based on a shared vision of the balance, continuity and governance of the care services supported and meet the needs of those working at all levels of health care in providing them. It must, as well, enable and support citizens acting in support of their own health care needs, and of those carrying caring burdens within families and local communities, focused on what matters to them. The citizen-facing information architecture of this new utility and the engineering methods employed in implementing and sustaining it will be fundamental to the successful reinvention and reform of health care. As I seek to demonstrate in Chapters Eight and Eight and a Half, there is already implemented and provenly applicable technology and method on which to base such a plan and platform, moving forward. It is growing widely across the world. Plans for technology promoting new scientific advance and related change and reorganization of services, must be aligned and adjusted around this core mission, with much greater attention to the radical change in services and society that may flow from them, and to proceed cautiously to secure benefits and avoid disbenefits.

In thinking about what can and should now be done along these lines, to help make the current situation better, we need first to understand why past failure to make progress, set against a backdrop of huge investments, has persisted for so long, and at such cost and expense. In a nutshell, it is because government and professional policy has championed an increasingly fragmented concept of health care, and information technology has accelerated this fragmentation. That is the nut of it, and

the shell is the championing of proprietary and fragmenting industrial models of information and information technology over the cultivation of a coherent common ground of requirement for information systems. This fragmentation persists across all the personal, professional, public and proprietary domains that must work together, coherently, in support of the balance, continuity and governance of cost-effective health care.

A patient may ask or expect their doctor to 'make me better'–in reality, it is a team effort, with the patient and their carers central to the team. A vituperative professor of surgery that I once trailed on his ward-rounds angrily rasped at his senior registrar, and delegated to him the task to 'get that patient well'! No amount of such rant helps anything get better but is the way of frustrated authority in difficult or chaotic times. Policy makers and other leaders have sometimes acted like that, implicitly, even if unknowingly so, delegating to engineers–way down stream, out of sight and mind–the task of realizing their dreams of future information for health. They have been frustrated that the levers they pulled, seeking to achieve their aspirations, proved unconnected with the outcomes they wished for and expected. Lacking provenly implementable ideas for how to proceed, the pattern of recurrent illustrious reviews and policy resets, the papering over of past failure with promise (and re-promise) of new and better futures, dressed in new clothes that have seldom arrived on time, and when they did, often failed to fit–all this has led to repeated failure.

This deliberate caricature is all a bit polemical. But given the continuing and escalating, costly and burdensome implications of its underlying truths, it needs to be considered, reflected on and learned from–because it has not needed and does not need to be this way. In Chapter Seven, I track, in detail, fifty years of reports, policies, strategies, reorganizations and initiatives in health care IT. This reveals the accumulated depth of problems that have become entrenched in a hard to improve and hard to dislodge, legacy of rapidly obsolete information systems, inflexible to meet the evolving needs of medical science and health care. I have had my account of the scene reviewed by central actors of those times, who have not demurred. It is by no means only a problem of the health care sector. A forward-facing and evolutionary approach to creating a sustainable care information utility, to support the changing needs of health care, is put forward and further elaborated from Chapter Eight onwards in the book. To some eyes, it will, no doubt, appear a naively optimistic and inappropriate Dreaming.[59] But it is now a demonstrably tractable proposition, since it is already happening

59 On the Aboriginal concept of the Dreaming, see Preface.

and halfway there. I will now zoom out and cool down somewhat, to reflect on the wider societal context of all this.

The Information Age and the UK NHS arrived in tandem–the seventy-five-year evolution of information technology being mirrored in that of the NHS. They arrived in an era of upswing in society, reflected in a concern to address the causes of social deprivation and create a universal health care service. Robert Putnam characterized the sixty years of American society that followed from around the 1890s, as an upswing from 'I to we'. The subsequent sixty years, from around the 1950s, as a downswing from 'we to I'.[60] He charted these two eras using a wide range of timeseries population datasets, each exhibiting an upswing and following downswing, in the form of an inverted letter U (∩).

As the NHS arrived, UK society entered a similar transition from upswing to downswing. The parallel upswing of information technology has aligned with, and perhaps accentuated and powered, the fragmentation of the 'we' of community and the assertion of the 'I' of individualism that Putnam describes. Health care services swung to an industrial and corporate managerial model of delivery, channeling new methods enabled by technology and new patterns of health care organizations and specialized professional services. From the mid-1960s, information technology sputtered into life, portended as a potent elixir–a solution to problems of supply and demand for boundless health care services. An echo of what the *Scientific American* journal said about the arrival of the motorcar, at the turn of the twentieth century, as I recalled in this book's Preface! The 2020s are poised at the limit of the Putnam downswing cycle, at an uncertain saddle point that can break, both up and down. Putnam is reassuringly positive that a new generation will bring energy and commitment to regeneration of upswing. There is much uncertainty, and much to make and do if this is to become a reality, just as in former and similarly uncertain times, as he recounts.

Befuddled by thwarted attempts to cope with the anarchic transition, government policies have been slow in recognizing the many and related integrative challenges of the fragmenting Information Age, and slow in adapting and evolving accordingly, to cope. Some of the disruption of health care services by information technology has been reminiscent of the battles over steam engines and steam power in the eighteenth and nineteenth centuries, as my examples in the chapter illustrated! It has been a costly anarchy of transition, tending to block the softer, more humble voices of experiment and learning which some that I have introduced in this book,

60 R. D. Putnam, *The Upswing: How America Came Together a Century Ago and How We Can Do It Again* (London: Simon and Schuster, 2020).

have represented, and argued for. The latest scientific or technological advances and buzzwords of organization have tended to be bandied with abandon, and successive reviews of past policies and future projections have mainly served to kick the can down the road. Genomics science, artificial intelligence and robotics have heralded an immense new adventure of ideas. They must be channelled in support of principled reinvention and reform of health care. This will open society to a new world, where a new Pandora's box of problems and concerns will, as ever, be poised to emerge. How we cope with and adapt to them, and what we make and do about them, is what will count.

It is as well to remember what David Graeber (1961–2020) wrote about the creation of the future, as quoted at the head of my Prologue. We all make the future and have choices in what we make, and how we make it, including about matters affecting our own health care. In these contexts, services are personal, and much information is personal. The quality of future health care services will reflect their enablement of all of us, as citizens, to become more instrumental, wherever possible, in making and doing much more of what we need and wish for, for ourselves and for one another, supported by related communities, professions, and institutions. The *how* of this will be common ground for all health care services, and a coherent and citizen-centred care information utility will be needed to support its emergent reality over the coming decades. This utility will need to look quite different from that created by the enclosure and commercialization of knowledge and personal data in the downswing era, if it is to help turn the inverted U (∩) world right-side up to U again, enabling and powering upswing in a new cycle of Putnam's 'I to we'.

The primary enabler of a successful care information utility will be trust on all sides, and trust is not something that can be hand-on-heart created; it is earned slowly and quickly lost. Like reputation, it arrives on foot and departs on a fast horse! The evolution towards a shared, sustained and trusted care information utility will be a slow one, but one that can start to be articulated, and gain traction, now. I have made a first, Aunt Sally-like, attempt in Part Three of this book. It will have multiple faces and must function securely and efficiently. It must span diverse locations and contexts, globally, while supporting balance, continuity and governance of services, locally. It must embody personal and professional co-ownership, social inclusion and community governance, tuned to the needs and expectations of all citizens of the Information Society. Who will make this happen, what will they do, where and how? There is a considerable legacy of existing stuff that will not do for this endeavour, and much new stuff taking shape to help us make and do things better, now. Progress in this direction is emerging in many countries–in Europe, for example, in Sweden, Norway, Spain,

Slovenia and Finland, including in whole health economy programmes of reform that I have been observing and engaging with.

An architectural blueprint is needed, covering all bases of the who, what, when, where, how and why of principled mission, goal and plan for the utility. Such a mission has proved beyond the individual scope and capacity of governments, professions, services, and industries, especially in large health economies. It is a community interest and will need to embody all of these participants, but also to be created afresh on new common ground, with citizen focus and ownership, and that not just in name. I have characterized this community interest and mission as 'care information utility with you in charge'–palindromic CIU with uic! ! More catchily, perhaps, as openCare! The primary users will be individual citizens in partnership with the professionals who serve and support them. Its scope will embrace both health and social care, signifying their joint identity and chemistry, held together in a rewarding exchange between the two domains. This characterization suggested itself to me when reading descriptions, today, of the covalent organic frameworks of chemistry.

Part Three of the book starts to articulate a Dreamtime-like vision of the creation of the future openCare utility. Before addressing the challenge of the future, it is necessary to describe and seek understanding of the past and present reality. In that regard, Chapter Seven in Part Two includes a good deal of critical commentary, but I hope it is fair. Part Three is an optimistic perspective which I likewise hope is not too starry-eyed. I have listened to and been guided by my colleagues closest to the health care service realities of recent decades, who have mentored me in both aspects.

One final personal reflection at this point, from one now long retired from the everyday fray of working life. There is a clearly identified group in society who will always have a special interest in the coherence and continuity of the services that the care information utility will support, being typically more aware of and engaged in their own side of the bargain in keeping well. This is the community of lucky and able, young-at-heart, retired citizens, who often seek and need new human connections through which they can feel needed and valued, for a hoped-for ten to twenty years of not-so-busy but still healthy and active retirement. There could be an appealing and win-win opportunity to articulate and support recognized roles for this group. They are present in families and communities everywhere, and are well-placed to help create, populate and operate a coherent common ground of care information utility. They have much to offer in this way for the reinvention of health care, by sharing their varieties of experience and skill, and, as importantly, their time and concern. I like the idea that care of the elderly might in this way have a reflection in the elderly of care!

Of course, very many citizens, and certainly not just the retired, have always contributed to care, in their everyday personal lives and the voluntary sector, as well as in professional careers and roles. The re-invention of health care for the Information Society must seek to better recognize and draw together all who provide care. Roll on openCare!