The automatic production of space

Nigel Thrift* and Shaun Frenchꞇ

This paper is concerned with the changing nature of space. More and more of the spaces of everyday life come loaded up with software, lines of code that are installing a new kind of automatically reproduced background and whose nature is only now starting to become clear. This paper is an attempt to map out this background. The paper begins by considering the nature of software. Subsequently, a simple audit is undertaken of where software is chiefly to be found in the spaces of everyday life. The next part of the paper notes the way in which more and more of this software is written to mimic corporeal intelligence, so as to produce a better and more unobtrusive fit with habitation. The paper then sets out three different geographies of software and the way in which they are implicated in the reproduction of everyday life before concluding with a consideration of the degree to which we might consider the rise of software as an epochal event or something much more modest.

key words software code timespace everyday life automatic

*School of Geographical Sciences, University of Bristol, Bristol BS8 1SS
email: N.J.Thrift@bristol.ac.uk

ꞇSchool of Geography, University of Nottingham, Nottingham NG7 2RD
email: Shaun.French@nottingham.ac.uk

revised manuscript received 6 February 2002

> [The] co-implication of discourse and referent is more than metaphorical because the linguistic system is not separate from the world, but as technology, as an articulation of life, is a natural extension of it. (Johnson 1993, 199)

> One thing is for sure. In the domain of the text, imagination is king. But a king subject to laws, laws of the story. (Appelbaum 1995, ix)

Introduction

This paper is an attempt to document the major change that is taking place in the way that Euro-American Societies are run as they increasingly become interwoven with computer 'software'.[1] Though this change has been pointed to in many writings, it has only rarely been systematically worked through. This, then, is the first goal of this paper – to systematically register this change and its extent. We hope to show how, in only 50 years or so, the technical substrate of Euro-American societies has changed decisively as software has come to intervene in nearly all aspects of everyday life and has begun to sink into its taken-for-granted background. But simply registering this change is not enough. Our second goal is to explain why it has been so effective. And here we point to software's ability to act as a means of providing a new and complex form of automated spatiality, complex ethologies of software and other entities which, too often in the past, have been studied as if human agency is clearly the directive force. In other words, what we believe we are increasingly seeing is the automatic production of space, which has important consequences for what we regard as the world's phenomenality, new landscapes of code that are now beginning to make their own emergent ways.

But it has to be said that the description 'software' is not an easy one to work with because, in the literature, all kinds of different meanings of

ISSN 0020-2754

'software' are routinely conflated, with the result that different kinds of efficacy are muddled up. Thus, at the most general level, software is often considered to be part of a more general structure of writing, a vast Derridean intertext that has gradually become a system without edges and that includes all manner of 'coded' writings rooted in a base cybernetic metaphor (Johnson 1993; Hayles 1999; Kay 2000). In such a conception, software is both a measure of how writing is now done, and a new kind of cultural memory based upon discourses of information as pure digital technique (Hobart and Schiffman 1998). In a second guise, software can be considered as another step in the history of writing as a supplement to spoken language: here we have Derrida's critique of phonocentrism and logocentricsm made flesh (Derrida 1998; Aarseth 1997).

> An ever-increasing number of people are spending more hours per day using written – that is, keyboard – language rather than spoken language. Within a few years, computers will be enriching nearly every household of the developed world. Human life in these countries is centering on, and contracting to, electronic text and international networking, and moving away from speech. Soon written language might be more prominent worldwide than spoken language. A different sort of language is emerging from this artificial interfacing: an 'oral-written language' occupying a special position between spoken and written language. Computers now regularly communicate with one another, too, through writing – that is, through written programming languages – without human mediation. Writing has, in this way transcended humanity itself. We have redefined the very meaning of writing itself. (Fischer 2001, 316)

Within this general shift, software can be thought of as a set of new textualities: programming languages, e-mail and other forms of 'netspeak' (Crystal 2000), and software packages, each with their own textual protocols and paratexts, which have produced their own linguistic turn. Then, in a third guise, software can be thought of as the product of the actual writing of code, as the outcome of the 'practised hands' (McCullough 1998) of a comparative handful of people who are able to mobilize skills that even now are difficult to describe to produce effective forms of code (Lohr 2001). Such skilful interaction between humans and machines has been the object of numerous studies in the human–computer interaction (HCI) and computer-supported cooperative work (CSCW) literatures, which all show that there is no straightforwardly observable exchange between discrete purified entities called 'human' and 'machine', but rather a series of conversations which demonstrate that software is not a simple intermediary, but rather a Latourian 'mediary' with its own powers (see, for example, Thomas 1995). Then software has one more guise. It can be couched as the guts of a set of commodities: websites, software packages, games, animated movies, and so on, which are dispensed via the medium of the screen and have become part of a more general cultural ambience. The ubiquity of the screen (see McCarthy 2001) guarantees software's cultural hold, letting it assume a central economic and cultural place.

Whatever the guise, software clearly stands for a new set of effectivities. But, of course, such a statement begs a whole series of important questions. For example, how does software relate to hardware? Is it, as in Kittler's (1997) extreme reaction, the case that there is no software, only hardware? Or is it that the hardware is now secondary? And, as another example, given that software has a hold on the world, what is the exact unit of efficacy? A line of code? An algorithm or a program? A complete 'informational ecology' (Nardi and O'Day 1999)? And, as one more example, is software's address now somewhere between the artificial and a new kind of natural, the dead and a new kind of living, or the material and a new kind of material semiotics? These are not questions we can hope to answer fully in such a short paper but, hopefully, we can start to open them out, for what is certain is that software is displacing some of our fondest conclusions about our contemporary age and cultural idiom as it provides us with 'new and different means of informing' (Hobart and Schiffman 1998, 268). We will attempt this opening out through a paper that is structured in five parts. In the first part, we will consider the nature of software. Our argument here is that software consists of a series of 'writing acts' which have changed our expectation of what can show up in the everyday world. This definitional section having given software a kind of voice, we can then move on to consider through a simple audit – no more, no less – how that voice is increasingly heard in everyday life as software achieves presence as 'local intelligence'.

Then, third, we will consider the ways in which all this software has been transmuting and, in

particular, how it is beginning to recognize and play to apparently human characteristics like emotion and is therefore starting to take on some of the characteristics of corporeal intelligences (just as corporeal intelligences are beginning to take on some of the characteristics of software; Collins and Kusch 1998). In the penultimate section, we will attempt to touch on the 'absent presence' of software in everyday life, by outlining three different geographies of software: (1) how and where software is produced; (2) the rise of new informational standards of conduct; (3) new forms of creativity and play. In the conclusion, we will try to sum up and extend our narrative by considering software as a new form of gathering, what we might call 'semi-artificial life' (Borgmann 2000).

The machinations of writing

In this section, we want to begin our account of software by briefly considering its nature. In doing so, we have to face up to the fact that software is often considered to be a 'technology', and is then able to be negatively assimilated into the domain of thought as a figure or metaphor representing some social else, thus allowing it to be captured within a linguistic or semiotically contrived field, rather than disclosed as an agent of 'material complexification' (Hansen 2000). But software cannot be reduced to this kind of textualism, even though it can be regarded as a text. That would be to domesticate its generative alterity, to simplify what can be regarded as intent, and to draw a veil over a politics of the shifting boundaries defining 'objects' which is so crucial to the contemporary 'human' sciences (Oyama 2001).

For a long time, much of the human world has been on automatic, has expanded beyond the immediate influence of bodies and has made its way into machines. The expansion of humanity beyond bodies has taken place in two ways, as a result of the invention of writing and then print, and as a result of the invention of various machines; line-by-line instructions and rude mechanicals. 'Software' and 'hardware'. In the past, these two means of manipulating the world have often been held separate. But now what we are seeing is an age in which writing is able to take on many new mechanical aspects – what we are seeing coming into being, therefore, is an age of software, but software becoming so pervasive and complex that it is beginning to take on many of the features of an organism.[2] But this is an organism with a passion for inscription, which goes to show that 'the logocentric repression of writing is to an extent only now visible and understandable in the light of recent developments in contemporary science' (Johnson 1993, 191).

We believe that this gradual evolution of software into what Clark (2000) calls 'wideware' is extraordinarily important for understanding the current direction of Euro-American cultures, and especially the nature of Western cities. Increasingly, spaces like cities – where most software is gathered and has its effects – are being run by mechanical writing, are being beckoned into existence by code. Yet, remarkably, this development has gone almost unrecorded. Why so? There are four immediate reasons, we think.

First, software takes up little in the way of visible physical space. It generally occupies micro-spaces. Second, software is deferred. It expresses the co-presence of different times, the time of its production and its subsequent dictation of future moments. So the practical politics of the decisions about production are built into the software and rarely recur at a later date. Third, software, is therefore a space that is constantly in-between, a mass-produced series of instructions that lie in the interstices of everyday life, pocket dictators that are constantly expressing themselves. Fourth, we are schooled in ignoring software, just as we are schooled in ignoring standards and classifications (Bowker and Star 1999). Software very rapidly takes on the status of background and therefore is rarely considered anew.

It would be easy at this point to fall back on some familiar notions to describe software's grip on spaces like cities. One would be hegemony. But that notion suggests a purposeful project, whilst software consists of numerous projects cycling through and continually being rewritten in code. Another notion would be haunting. But again the notion is not quite the right one. Ghosts are ethereal presences, phantoms that are only half-there, which usually obtain their effects by stirring up emotions – of fear, angst, regret, and the like. Software is more like a kind of traffic between beings, wherein one sees, so to speak, the *effects* of the relationship. What transpires becomes reified in actions, body stances, general anticipations (Strathern 1999). We would argue, then, that software is best thought of as a kind of absorption, an

expectation of what will turn up in the everyday world. Software is a new kind of phenomenality which can 'touch the ontic' (Spivak 1993, 30). Software is, in other words, a part of a 'technological unconscious' (Clough 2000), a means of sustaining presence which we cannot access but which clearly has effects, a technical substrate of unconscious meaning and activity. 'It is, after all, against the natural unity of self-heard voice that Derrida places technicity, the machine, the text, writing – all as bearers of unconscious thought' (Clough 2000, 17). Increasingly, therefore, as software gains this unconscious presence, spaces like cities will bear its mark, bugged by new kinds of pleasures, obsessions, anxieties and phobias which exist in an insistent elsewhere (Vidler 2000; Thrift 2001). Software quite literally conditions our existence, very often 'outside of the phenomenal field of subjectivity' (Hansen 2000, 17).

Thus we come to a more general reason why software remains so little considered. We still cleave to the idea of spaces like the city as populated by humans and objects that represent each other via words and images, which makes it very difficult to mark this new territory (cf. Downey and McGuigan 1999). Software does not fit this representational model, the 'theatre of proof' (Phelan 1993), for its text is about words doing things, about determinate presentations in particular contexts 'below the ''threshold'' of representation itself' (Hansen 2000, 4). Because software 'affects our experience first and foremost through its infrastructural role, its import occurs prior to and independently of our production of representations' (Hansen 2000, 4). Seen in this way, software is perhaps better thought of as a series of 'writing acts' (rather than speech acts) of a Bakhtinian or Derridean kind, which have a 'heuretic', rather than an analytical dimension (Ulmer 1989), based upon the inventive rather than the analytic, in which language is both message and medium. Thus

> Within the previous instauration, founded on the alphabet, the only way to access theory *per se* was through metaphor, every concrete manifestation of the idea being equivalent to its deformation. Metaphors were necessary because the intellect was otherwise incapable of grasping the idea of true illumination. With the new instauration, the artefacts of theory are no longer metaphors. Instead, the object is no longer the deformation of the idea, but is its real embodiment. Now the idea, or thought, rests within, or out of, the object itself. (Lechte 1999, 141)

It then becomes something of a moot point whether this means that software – as a non-representational form of action – 'does not rely on the activity of thinking for its ontogenesis' (Hansen 2000, 19) or whether it is simply another kind of distributed thinking in which yet more human functions are delegated into 'the automatic, autonomous and auto-mobile processes of the machine' (Johnson 1999, 122), as part of a process of externalization and extension of the vital based, for example, on the apprehension of the human body as simply 'too slow' (Stiegler 1998). Whichever the case, what we can see is that what counts as 'life' itself comes into question as new material syntheses emerge and embed themselves (Doyle 1997).

In the rest of this paper we will therefore chiefly call on two theoretical technologies to understand software. One is a whole series of theoretical elements that are now merging which, when taken together, constitute a 'theatre of promise' in their emphasis on the how of practice, their attention to hybrid human–object networks (cyborgs, actants, ethologies, and the like) and their devotion to the task of repopulating the spaces of modernity with these new inhuman figures. The other is the emphasis on the performativity of writing. Drawing on work as diverse as Derrida and performance studies, this other 'theatre of promise' constitutes a series of plays with scriptural space and time, which are intended to extend the nature of writing whilst retaining its 'prophetic touch' (Johnson 1993 1999).

Mapping software

To begin our audit of software's locations, let us try to understand just how much things have changed by journeying back to the 1970s. In a remarkable and unjustly neglected paper, Ron Horvath argued that much of the interior of Western cities had become a new kind of machine wilderness, which he called 'machine space'. He saw this new territory, 'devoted primarily ... to the use of machines' (Horvath 1974, 167), as a desolate and threatening one, because it gave priority to machines over people. Horvath was writing chiefly of the automobile and he mapped out in detail the 'expanding realm of machinekind' (Horvath 1974, 187) by paying attention to the increasing amount of space being given over to cars in American cities

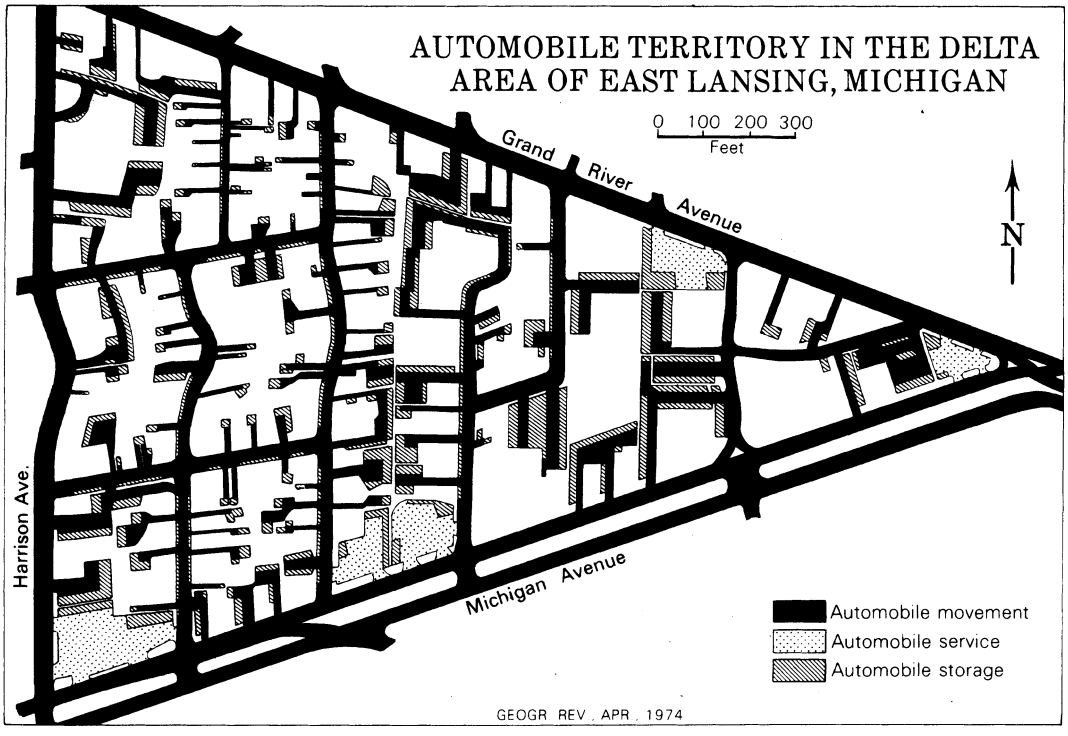

Figure 1 Machine space (Horvath 1974)

(see Figure 1) and the consequent deathly effects. Yet now we see something of the same kind of machinic expansion occurring on as great a scale (Mackenzie 1996). But even though software has infused into the very fabric of everyday life – just like the automobile – it brings no such level of questioning in its wake.

Indeed, the automobile is a good place to start our audit, since automobiles are increasingly stuffed full of software. Reporting on the explosion of electronics within motor cars, the Economist Intelligence Unit (EIU 2000) predict, for instance, that within a very short space of time electronics will constitute more than 30 per cent of an executive car's total value. Such vehicle electronics are increasingly driven by smart software, developed by individual companies and through partnerships such as the Intelligent Vehicle Initiative (IVI) in the United States. Work within the IVI program includes the development of intelligent collision avoidance systems; traffic, location and route guidance; driver condition monitoring systems; and obstacle and pedestrian detection systems. As we write, Ford UK is currently marketing its Mondeo model on the basis of its Intelligent Protection System (IPS). A neural network of sensors embedded within the Mondeo enable IPS to assess the severity of any impact, instantaneously adjusting the car's safety systems to maximize their effectiveness. Similarly, the new Jaguar X-type, based on the Mondeo platform, not only has a new familiar computerized navigation system, but also voice activation of some controls and a new feature called Jaguar Net, which integrates telephone and satellite location and automatically transmits an emergency call if there is an accident in which an airbag is deployed. The number of car components with these kind of embedded systems is such that an EIU report predicts that as early as 2001 passenger cars will appear on the market with 'vehicle intranets', connecting all the various onboard computers.

These intelligent vehicle programs are paralleled by a growing number of intelligent transportation initiatives. For instance, the US IVI program is integrated within a wider Intelligent Transportation System program. Drawing on developments in Australia, Japan and Europe, ITS

Table I Active Y2K websites

UK	General UK Y2K site. Includes downloadable copies of key Year 2000 reports	http://www.y2kbug.org.uk/
UK	British Computer Society Y2K Information	http://www.bcs.org.uk/millen.htm
UK	The Computer Information Centre Year 2000 Support Centre	http://www.compinfo.co.uk/y2k.htm
UK	BBC Education Millennium Bug Website	http://www.bbc.co.uk/education/cdb/bug/
UK	BBC News 'Bugtown' Y2K Website	http://news.bbc.co.uk/hi/english/static/millennium_bug/bugtown/
USA	National Y2K Clearinghouse (public, business and academic national resource)	http://www.y2k.gov/
USA	US Commercial Site	http://www.year2000.com/
USA	CBC News Y2K Site.	http://cbc.ca/news/indepth/y2k/

America (www.itsa.org), an amalgam of public and private bodies, seeks to foster the wider development, employment and integration of Intelligent Transportation Systems creating a more efficient, safer and more cost-effective transport network. Areas targeted by ITS include the collection and transmission of information on traffic conditions; the reduction of accidents and congestion; and the development of sophisticated navigation and route guidance systems for drivers. More specific examples of Intelligent Traffic Systems include the development of intelligent software for traffic lights. Employing fuzzy logic intelligent traffic lights are, for instance, able to read and react to changing traffic conditions, thus providing enhanced performance vis-à-vis conventional fixed-time controlled lights (Khiang Tam *et al.* 1996).

But it is easy to move beyond automobiles and roads in the search for urban software. Elevators are another integral part of the infrastructure of cities that have become increasingly software rich. From being held up as an exemplar of a 'stupid machine' in the early 1980s, James Gleick points out that elevators have now surpassed even the motor car in terms of their software richness. According to Gleick, smart elevators now pack 'more computing power than a high-end automobile, which is to say more computing power than the Apollo spacecraft' (Gleick 1999, 24). By adding microprocessors, again programmed with fuzzy logic, smart elevators or 'elevators with algorithms' have:

> learned to skip floors when they are already full, to avoid bunching up, and to recognise human behaviour patterns. They can anticipate the hordes who will gather on certain floors and start pounding the DOWN [sic] button at 4.55 p.m. each Friday. (Gleick 1999, 24–5)

The infusion of software into the urban infrastructure has not only impacted upon older urban technologies, it is also a vital part of newer technologies. Despite only being a relatively new addition to the fabric of cities, security surveillance or CCTV systems are in the process of being revolutionized (Graham 1998). In October 1998, the London borough of Newham became the first place in the world to employ smart CCTV to monitor public spaces (Kinnes 2000). Developed by Visionics, the Faceit surveillance system uses cutting edge facial recognition technology to scan for criminals. Faceit is able to do this by 'mimicking the brain, the software scans and measures distances between the major landmarks of the face, reducing them to a heap of pixels' (Kinnes 2000, 14). Following Newham, a similar anti-crime smart CCTV system, Satnet, has been introduced to the West End, while Cromatica, designed to monitor crowd flows, congestion and prevent suicides, has been tested on the London Underground (*New Scientist* 1999a 1999b 2000).

Is there any way of making a more general assessment of software in the city? Conveniently, the millennium bug (Y2K) has provided a close approximation to such a general audit.[3] A brief examination of the UK Audit Commission's (1998) report on Y2K immediately brings into sharp relief the degree to which software and related embedded systems have infused everyday urban life. In addition to traffic lights and lifts, the Audit Commission lists car park barriers, central heating boilers, building security systems, burglar and fire alarms, accounting software, vehicle fleet maintenance systems, local authority revenue systems, child protection registers, benefit systems, emergency service communication systems and medical equipment, as just some of the areas with the potential to seriously disrupt everyday life as a

consequence of their reliance on embedded and computerized systems. Other notable landmarks of 'Bugtown UK' (Figure 2) included the banking industry, gas, water and electricity supply, food retailing, the post office, the police force, the fire brigade, hospitals and emergency services. In the case of hospitals, the Audit Commission used the following passage to warn of the extent of the potential problem:

> The year 2000 project manager at one NHS trust is quoted as saying: 'The [potential] problem extends to all areas of the hospital – lifts, diagnostic equipment, X-ray machines, anaesthetics, breathing equipment and monitors'. (Audit Commission 1998, 11)

The Year 2000 problem was a global problem, and similar kinds of audits were carried out all over the world. For example, Table II lists the compliance issues identified by the City of Beverly in Massachusetts.

In the event, of course, the disruption caused by the 'millennium bug' was negligible and there have since been questions raised as to how much of a true threat Y2K ever posed. Nevertheless, if it achieved nothing else, Y2K served to amply illustrate the extent to which software has seeped into everyday life.

If, up until now, much of this infusion of software into everyday life has gone largely unnoticed, newer technological developments will, by virtue of their closeness to bodies, be much more readily apparent (though not necessarily visible). This is the world of 'local intelligence' in which everyday spaces become saturated with computational capacities, thereby transforming more and more spaces into computationally active environments able to communicate within and with each other. This change is taking place as a result of two main developments. The first of these is the move to 'ubiquitous', 'everywhere' or 'pervasive' computing, computational systems which are distributed through the environment in a whole range of devices, 'a physical world invisibly interwoven with sensors, actuators, displays and computational elements, embedded seamlessly in the everyday objects of our lives and connected through a continuous network' (Weiser *et al.* 1999, 2–3).

This is the world of 'information appliances' foreseen by Norman (1998) in which each of us will, in a sense, put our own computing needs together as and when necessary. 'Information appliances should be thought of as systems, not isolated devices' (Norman 1998, 253) so that 'instead of one massive device that occupies considerable space on our desk top, we will have a wide range of devices that are designed to fit the tasks that we wish to do' (Norman 2000, 12). Today, of course, we only have access to the first generation of appliances meant to inform everyday life, devices which still too often bear the mark of 'computing', chiefly personal digital devices (like the Palm Pilot), mobile phones, recordable CD players, portable MP3 players, personal voice recorders, interactive pagers, internet radios, and so on, which have some computing capacity and can often communicate with each other. But increasingly, this kind of computing will likely make its way out into many more kinds of devices and become personally tailored (see Bergman 2000). This is a move away from an internet-inspired idea of computing as concerned with the provision of specific stand-alone high technology devices that provide analyses and representations of information to computing as distributed through a whole series of devices used as and when – and where – appropriate. 'Less fuss and bother. Simpler, more convenient devices. Great flexibility and versatility. New modes of interaction, of learning, of conducting business and recreation' (Norman 1998, 261).

The second development is the move to position the internet – the obsession of the 1990s – as but one element of a computing revolution:

> the use of the Internet [will be] so pervasive, so natural, and so commonplace that the very notion of calling something an 'internet appliance' will be completely unnecessary. (Norman 1998, 269)

Rather we will exist in a broadband world in which the internet will be a permanently available 'cloud' of information able to be called up through a number of appliances scattered through the environment. These appliances will be something more than portals for information. Rather, like many other devices which will not have internet connections, they will be 'practice-aware', responding to and aware of the context in which they are used through an array of wireless and other sensors, continuous locational information read from Global Positioning System (GPS) references, and the like. In turn, this means we are moving away from machines that simply respond to machines that interact because they are aware

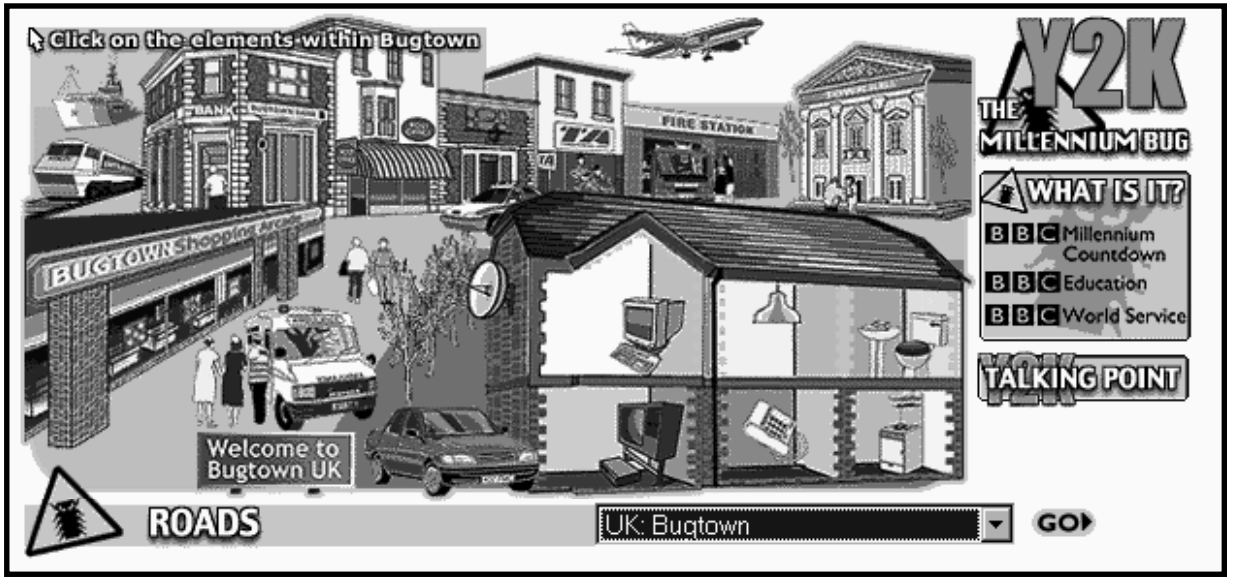

Figure 2 'Bugtown UK' – the BBC's Y2K information site
Source: http://news.bbc.co.uk/hi/english/static/millennium_bug/bugtown/

Table II Some of the building and equipment issues included in the city of Beverly's Y2K compliance audit of local government services

Buildings	*Building security and safety*	*Environmental*	*Communications*	*Office equipment*
Building access	Security alarms	Electronic heat control/thermostats	Phone service	Copy machines
Electronic locks	Fire alarms	Electronic air conditioning control	Phone switch	Fax machines
Card keys	Fire suppression/sprinkler	Heat plant	Voice mail system	Postage meter
Keypad locks	Fire door controls	Air conditioning plant	Local carrier	Electronic scales
Time locks	Alarmed crash bars	Fuel source	Long distance carrier	Computer terminals
Handicap lifts	Environment alarms	Energy management	Cell phone	Desktop computers
Elevators	Carbon monoxide detectors	Lighting management	Cell phone service	Laptop computers
Intercom systems	Radon detectors	Backup generators	Pagers	Printers
Entry buzzers	Natural gas detectors	Water/sump pumps	Pager service	Personal digital assistant
Parking garage gates	Water alarms	Water meters	Radios – 2 way	Specialized software
Automatic garage doors	Storage tank alarms	Solar panels	Radios – mobile	Scanners
Closed circuit entry cameras	Emergency lights		Radios – marine	VCR
	Emergency light batteries		Radio service	Camcorders
	Battery chargers			Tape recorders
	Security rounds clocks			Digital cameras
	Security cameras			
	Security monitors			
	Videotaped surveillance			
	Recorded phone system			

Source: City of Beverly Y2K Audit Committee Report (1998), Massachusetts, United States

enough of the context in which they operate to be able to do so. This means more than simply better human–machine interfaces (though these will be a part of this new interactivity (Johnson 1997)). And it means more than simply the redesign of the overall encounter with a machine, so that it becomes a more satisfying experience (see Dryer *et al*. 1999). Rather it means that a whole set of appliances (which 'compute') will, through a process of cultural absorption into practices, sink down from the representational into the non-representational world, so becoming a part of a taken-for-granted set of passions and skills (Thrift 2000).

At present, this world of local intelligence is still some way off. But not that far off. Two examples will suffice to make the case: mobile phone technology and so-called 'wearables'. With the advent of third generation (3G) technologies, the software running mobile phones and mobile phone networks is becoming ever more sophisticated such that, just like the car or the elevator, the mobile is set to become 'smarter' (*New Scientist* 2000). Mobile phones incorporating speech recognition technology to allow hands-free dialling and access to functions are already in common use. In addition, work is underway on installing 'Bluetooth' and other wireless communication chips, in addition to SIM cards, into 3G phones. Such chips will allow communication with other Bluetooth devices within the immediate vicinity (roughly a house-shaped bubble) of the mobile phone user. This would allow, as one simple instance, the retailers in city centres and shopping malls to send a '''10 per cent off'' coupon to your mobile's screen as you approach' (*New Scientist* 2000, 33). Following on from this kind of approach, it is clear that messages soon will be pinned in mid-air using GPS.

> Messages [will not] actually be kept in the air; they're stored on a web page. But that page's Web address is linked to co-ordinates on the Earth's surface, rather than the organisation. As you move about, a GPS receiver in your mobile phone or PDA will check to see what has been posted on the website for that particular spot. If you're in luck a snippet of info left as a recording by someone who passed there previously will pop up on your screen or be whispered into your earpiece. (*New Scientist* 2001, 16)

Already, prototype systems like Hewlett Packard's COOLTOWN (http://cooltown.hp.com) are attempting to produce spaces in which universal open messaging is commonplace.

The way in which mobile telephony has already become a part of everyday life, producing new forms of social action – from the new kinds of 'hyper-coordination' promoted by text messaging (and the new kinds of 'flocking' that is made possible) to the invasion of public space by private and work lives to new kinds of affective social performance – shows, in turn, the way that even very basic forms of local intelligence can have substantive cultural effects. (See Townsend 2000; Laurier and Philo 2001; Brown *et al*. 2002).

A second example of nascent local intelligence is provided by wearable computing, which has begun to develop strongly in the last five years as a means of providing computing that is always ready and available by virtue of being present in items of clothing: 'it's always on, it's always accessible and it's always part of you' (Billinghurst n.d.). 'It therefore exists within the corporeal envelope of the user' (Bass 1997, 23). Though commercial wearables will at first consist of items like Levis IGD+ (developed in partnership with Phillips and featuring the Xenium speech recognition phone), which are little more than bulky multi-pocketed jackets able to contain various pieces of electronic kit, the future might be very different. Cyberjackets will probably be the first step. Such cyberjackets will, for example, be able to alert and guide a wearer to interesting shops. 'Shopping Jackets' are, however, only one of a whole host of potential wearable applications developed to transcend the conventional user/PC paradigm. For example, the Belgian company Starlab have unveiled layers of i-Wear with memory, communication, interface, power, connectivity and computational abilities (www.starlab.org).

More recently, it has become possible to create computationally active textiles, which are able to weave circuit substances into cloth, so creating the possibility of ever more corporeally sensitive interfaces (Post *et al*. 2000). By installing computing systems into jackets, trousers, hats, shoes, glasses, and the like, wearable pioneers like Mann have sought to radically reconfigure the way in which we use computers. Smart clothing transforms computation into 'intelligent assistance', actively rather than passively engaging with the user (MIT website: www.wearcam.org/computing.html/).[4] Thus, wearables can not only assist in locating shops, they can also act as more general navigation aids, as mobile payment systems, provide security

access to buildings, assist engineers and mechanics in the field, record conversations, meetings and other events, act as mobile internet and phone portals, augment vision and memory and a host of other activities. And, as in the case of mobile phones, a key requirement of future wearable systems will be the need to communicate, through the employment of wireless protocols like Bluetooth, with each other and with other systems embedded in the fabric of everyday life.

The trend towards local intelligence will hardly stop at mobile phones and wearables, as a glance at any journal like *Personal Technologies* shows. Computational ability is being embedded in household whitegoods, in furniture (including beds and couches), even in carpets (Omojola *et al*. 2000; Paradiso *et al*. 2000).

However, it is also important to note that some of the most important imprints of local intelligence will come from the increasing ability afforded by Bluetooth and similar wireless protocols to produce a multitude of 'invisible' embedded systems, communicating not with human users, but with each other, so producing a genuine 'mechanosphere'.[5] In certain contexts it may even be, as *New Scientist* (2000, 33) predicts, that communication between people will become subservient to machine-to-machine communication. So, for example, 'the number of phone calls between people will be overtaken by machines talking to machines on behalf of people' with the result that

> [a]s soon as your washing machine is installed, it will be on the air to your Bluetooth controller, asking if it can contact its manufacturer over the Net. A year later you'll trip over a repair engineer who's been e-mailed by the washing machine because it has a worn bearing. (*New Scientist* 2000, 33)

In concluding this section of the paper, it is important to note that this kind of actual and prospective audit should not give the air of a kind of technological finality, for two reasons. First, the new generation of intelligent devices are already being socially and culturally inflected in quite different ways in different cultures. A good current example is the radically different pattern of use of mobile telephony and PDAs in Europe and North America. Quite unexpectedly, different practices have already grown up on both sides of the Atlantic in relation to the take-up and use of these two technologies (see Brown *et al*. 2002). Then, second, software itself hardly constitutes a smoothly functioning and well-versed set of procedures. This is a point worth expanding. To begin with, software consists of many, often incompatible, languages, which are constantly changing.

> I learned to program a computer in 1971; my first programming job came in 1978. Since then, I have taught myself six higher level programming languages, three assemblers, two data-retrieval languages, eight job-processing languages, seventeen scripting languages, ten types of macros, two object definition languages, sixty-eight programming library interfaces, five varieties of networks and eight operating environments – fifteen, if you cross-multiply the different combinations of operating systems and networks. I don't think this makes me particularly unusual. Given the rate of change in computing, anyone who's been around for a while could probably make a list like this. (Ullman 1997, 101)

Then, software is built up from many different components, many of which are 'legacy systems' that have existed for many years. In effect, many different programmers still live in the code (Downey 1998).

> Software gets to age. Too much time is invested in it, too much time will be needed to replace it. So unlike the tossed-out hardware, software is tinkered with. It is mended and fixed, patched and reused . . . I once worked on a mainframe computer system where the fan-folded history of my COBOL program stood as high as a person. My program was sixteen years old when I inherited it. According to the library logs, ninety-six programmers had worked on it before I had. I spent a year wandering its subroutines and service modules, but there were still mysterious places I did not touch. There were bugs on this system no one had been able to fix for ten years. There were sections where adding a single line of code created odd and puzzling outcomes programmers called 'side effects': bugs that come not directly from the added code but from some later, unknown perturbation further down in the process. My program was near the end of its 'life cycle'. It was close to death.
>
> Yet this system could not be thrown away. By the time a computer system becomes old, no one completely understands it. A system made out of old junky technology becomes, paradoxically, precious. . . .
>
> The preciousness of an old system is axiomatic. The longer the system has been running, the greater the number of programmers who have worked on it, the less any one person understands it. As years pass and untold numbers of programmers and analysts come and go, the system takes on a life of its own. It runs. That is its claim to existence: it does useful work.

> However badly, however buggy, however obsolete – it runs. And no one individual completely understands how. (Ullman 1997, 116–17)

In other words, programmers function against a background of 'ignorant expertise'.

> The corollary of constant change is ignorance. This is not often talked about: we computer experts barely know what we are doing. We're good at fussing and figuring out. We function well in a sea of unknowns. Our experience has only prepared us to deal with confusion. A programmer who denies this is probably lying, or else is densely unaware of himself. (Ullman 1997, 110)

No wonder that most software is only properly tested out through a process of trial and error in real applications (Mackenzie 1996).

Again, as numerous writers have pointed out (e.g. Winograd 1996; Norman 1998), software is rarely designed well, so that the capacities of most programmes are underused:

> One of the main reasons most computer software is so abysmal is that it's not designed at all, but merely engineered. Another reason is that implementers often place more emphasis on a program's internal construction than on its external design, despite the fact that as much as 75 per cent of the code in a modern program deals with the interface to the user. (Winograd 1996, 5)

Then, software does not always do as it is bidden by programmers or is used as it is meant to be by users. It is inescapably a joint production, a feature that is especially noticeable when the program is applied.

> Writing a software program . . . is a way of addressing a problem in an original manner. Each team of programmers redefines and resolves, although differently, the problem it is faced with. The actualisation of the program during use in a work environment, however, ignores certain skills, reveals new kinds of functionality, gives rise to conflicts, resolves problems, and initiates a new dynamic of collaboration. The software carries with it a virtuality of change that the group . . . actualises in a more or less inventive way. (Lévy 1998, 25)

To conclude this section, one last point also needs to be made. The simple fact of the matter is that software, in the shape of embedded systems, is now so widespread that we are no longer able to be sure of its exact extent.[6] As the Audit Commission stressed in relation to the difficulties encountered in identifying Y2K-affected equipment, 'the presence of the embedded system may not be obvious *even to a trained observer* . . . [and some] systems may also be extremely difficult to locate or test' (Audit Commission 1998, 11, emphasis added). Yet the Commission estimated that even as far back as 1996, some 7 billion embedded systems were being distributed worldwide.

Breathing new life into software

Any audit of software in cities can therefore by its very nature only ever be partial and incomplete, and ours is no different. Indeed this recognition, more than anything else, illustrates just how extensive the distribution of this form of 'machine space' has become. But this may only be the beginning. Continuing developments within computer science suggest a much greater role for software than has so far been apparent as the machinic writing of software itself changes its form and capabilities.

Such change arises out of a general move away from formal well-specified programs towards programs that stress the situatedness of action, the importance of interaction and adaptation, and emergent properties. Such notions have existed for a long time now, spurred on by the ethnographic study of human–computer interaction and by more general developments in the social sciences and humanities (e.g. Suchman 1987). But, of late, these kinds of programs, using a diverse range of methodologies – from fuzzy sets to neural networks to generative algorithms to the data mining techniques of bio-informatics and bio-computation – have become much more prevalent (Bentley 2001). Here we note just one of these developments, the growth of the so-called 'soft computing' movement. Soft computing encompasses a range of methods that stress appropriate rather than precise models (Figure 3).

Fuzzy computation is a set of techniques most often associated with the work of Zadeh at the University of California, Berkeley from the 1960s on, on fuzzy sets, fuzzy logic and complex systems theory. In general terms fuzzy computation was born from a recognition that existing programming methodologies, which relied upon precise and detailed models, were inappropriate for dealing with complex, uncertain and vague systems or problems (Ruspini *et al.* 1998). Similarly, techniques of *evolutionary computing* grew out of a recognition of the limitations of precise modelling techniques for many empirical problems.[7] Drawing upon natural theories of 'reproduction, mutation and

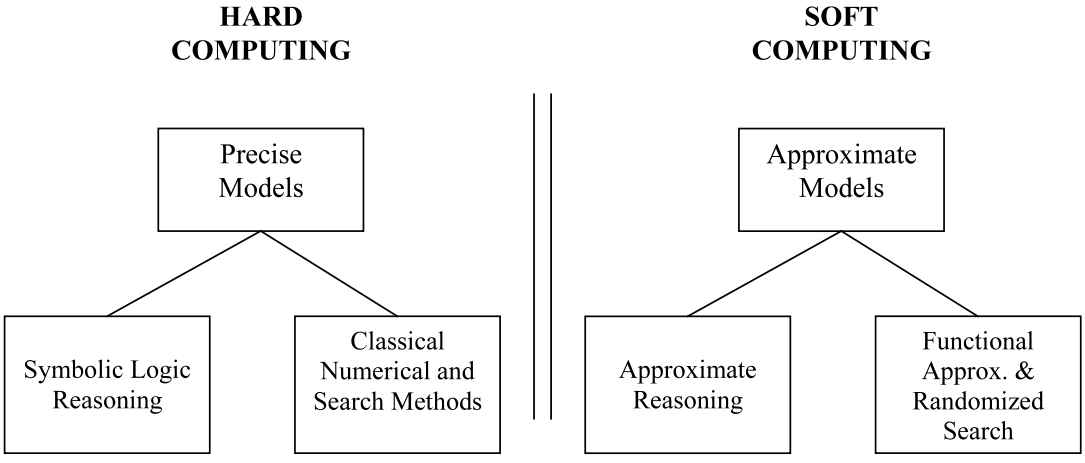

Figure 3 Hard and soft computing (Bonissone 1998)

the Darwinian principle of survival of the fittest', evolutionary computing seeks to harness the 'power of natural selection to turn computers into automatic optimisation and design tools' (http://evonet.dcs.napier.ac.uk/evoweb/; Heitkötter and Beasley 1999).

The term *soft computing* is closely linked to both that of fuzzy computation and evolutionary computation.[8] In attempting to define soft computing, Zadeh states that 'the role model for soft computing is the *human mind*' (1994, 27, emphasis added). Soft computing is distinguished by its tolerance of 'imprecision, uncertainty and partial truth' (Zadeh 1994, 27), in contrast with conventional hard computing with its emphasis upon 'crisp classifications' and perfect information. As Bonissone stresses:

> . . . when we attempt to solve real-world problems, we realise that they are typically ill-defined systems, difficult to model and with large-scale solution spaces . . . Therefore, we need hybrid approximate reasoning systems capable of handling this kind of imperfect information. (Bonissone 1998, D1.1:2)

Increasingly, many soft computing approaches have begun to share a general characteristic: a turn to biology and the natural sciences for inspiration. In particular, theories of evolution have inspired a host of alternative programming techniques, of which one of the most influential has been the genetic algorithm (GA). As with many other soft computing methods, GAs have provided the building blocks for some of the most sophisticated software applications, particularly artificial intelligence systems. It is therefore worth very briefly examining the rationale behind the genetic algorithm, both in its own right and as an exemplar of soft computing techniques.

The genetic algorithm was developed in the 1970s by Holland at the University of Michigan as a mechanism for addressing complex problems with very large 'solution spaces' (Sipper 2000; Holland 1975).[9] To tackle large solution space problems, Holland developed a 'model of machine learning which derives its behaviour from a metaphor of some of the mechanisms of evolution' (Heitkötter and Beasley 1999, 1). As with other types of evolutionary computing, GAs work by seeking to evolve solutions to a given problem. To achieve this, the GA employs a specific mode of *genetic representation* (see Table III). A population of

Table III Comparison of biological and GA terminology (Goldberg 1989, 22)

Biological	*Genetic algorithm*
Chromosome	String
Gene	Feature, character, or detector
Allele	Feature value
Locus	String position
Genotype	Structure
Phenotype	Parameter set, alternative solution, a decoded structure
Epistasis	Nonlinearity

individuals or bit *strings* of data, analogous to chromosomes in human DNA, is initially generated, each of which represents a possible solution. This first generation of candidate solutions is then evaluated in the context of its environment, its fitness to the problem in hand. On the basis of this fitness and the *genetic operator*[10] employed, a new generation of individuals is born (Goldberg 1989; Belew and Vose 1997; Heitkötter and Beasley 1999; Sipper 2000). By employing the principle of 'survival of the fittest' along with that of genetic diversity, each new generation produced should include solutions that are better than those of the previous generation. Of critical importance in understanding the power of the GA is that it removes the necessity of having a predetermined solution to a problem. Programmers simply have to determine an appropriate fitness function and genetic operator (Davis and Steenstrup 1990).

The genetic algorithm, along with the various other soft computing techniques, has been particularly important in the development of artificial intelligence (AI) systems. AI incorporates a wide variety of specific applications, from artificial life, robotics and recognition technologies through to data mining, expert systems and intelligent agents. One of the distinguishing features of all these various types of AI system is the fact that they operate within the very same, large solution spaces for which conventional hard computing approaches have shown themselves to be so ill-equipped. It is no surprise then that soft computational models have been mobilized to address such complex and indeterminate problems. Indeed much of the impetus behind the development of soft computing and the like has come from the desire to develop artificial life systems, biometrics and other smart technologies. This is little surprise since, beginning with the early days of cybernetics, there has been constant interchange between biology and computing as, for example, in the case of neural nets (Helmreich 1998; Anderson and Rosenfeld 1999; Hayles 1999; Oyama 2001).

Turning back to our audit, soft computation can be found in a great number of the embedded systems that populate cities. Intelligent traffic lights, elevators, automobiles and washing machines are programmed with fuzzy logic, as are many ITS systems (www.its.org). Genetic algorithms are helping to run medical diagnostic and monitoring equipment, data mining technologies, credit scoring and behavioural modelling systems, traffic management systems and call centre telephone routing technologies. More generally, genetic algorithms are being utilized in Artificial Life worlds such as Tierra, financial market models, flexible production systems, telecommunication networks and even by the UK's Channel 4 to assist in the scheduling of television advertisements (Helmreich 1998; Evonet: www.evonet.dcs.napier.ac.uk). Self-learning software is also being mobilized within the field of biometrics and in technologies of recognition (for example, in visual recognition systems).

These techniques have enabled machines and objects to begin to take on some of the characteristics of corporeal intelligence. Smart software has begun to breathe new kinds of life into a multitude of everyday things. This is not to suggest that software or, for that matter, computer systems have ever been dead (see Ullman 1997, 117; Downey 1998), but that the 'romantic reaction' (Turkle 1991, cited in Helmreich 1998, 140) to hard, rationalistic computing has helped software begin to take on many of the characteristics normally associated with biological life.

This new form of machine is no more starkly illustrated than in the case of a range of technologies designed to recognize and act upon the human body: the face, the voice, handwriting and, perhaps most remarkably, mood and emotion. The first and second of these, facial and speech recognition form part of a wider school of biometrics technologies, designed to recognize individuals from their distinguishing traits. Other biometrics technologies developed for a range of security uses include electronic fingerprinting, iris coding, hand geometry and palm print recognition. As we have shown, facial recognition technologies are already becoming a reality within cities, as are commercial voice recognition packages such as those developed by the Belgian company Lernout & Hauspie (L&H) and those incorporated within the Xenium mobile phone. L&H are not only developing speech recognition applications for use within call centres, consumer and business electronic packages, toys and wearables, but have also developed text recognition products.

Research institutes such as the MIT Media Lab have also conducted considerable work within the area of 'Affective Computing', computational systems with the ability to sense, recognize, understand and respond to human moods and emotions. Picard (1997 2000) argues that to be truly

interactive computers must have the power to recognize, feel and express emotion.

> Not all computers need to 'pay attention' to emotions or to have the capability to emulate emotion. Some machines are useful as rigid tools, and it is fine to keep them that way. However, there are situations in which human–computer interaction could be improved by having the computer adapted to the user, and in which communication about when, where, how and how important it is to adapt involves the use of emotional information. (Picard 2000, 1)

Plutowski (2000) identifies three broad categories of research within the area of affective or emotional computing (www.emotivate.com/Book/): emotional expression programs that display and communicate simulated emotional affect: emotional detection programs that recognize and react to emotional expression in humans; and emotional action tendencies, instilling computer programs with emotional processes in order to make them more efficient and effective. Thus, specific projects at MIT (www.media.mit.edu/affect/) have included the development of Affective Wearables such as Affective Jewellery, Expression Glasses, and a Conductor's Jacket designed to extend a Conductor's ability to express emotion and intentionality to the audience and to the orchestra. Other projects include: Affective Carpets; Affective Virtual Reality Avatars, which represent the changing emotional state of their users in the real world; Affective Toys, such as the 'Affective Tigger' and Orpheus; and an Affective CD player, which selects music in alignment with current mood.

Affective computing is already making its way into the commercial world. Basic emotional expression and detection technology is being employed through the use of 'embodied conversational agents' (see Cassell *et al*. 2000; Pesce 2000), icons which are able to mimic conversation, not only through better linguistic skills, but also through employing a range of the non-verbal behaviours associated with affect like facial expression, gesture and posture. Such agents therefore operate not only in the representational register (as in, for example, representing concepts), but also in the non-representational register (as in spatializing or providing rhythm to a conversation). Such icons now encompass a whole range of agencies, including not only roles in games, but also intelligent avatars like virtual pets and virtual friends, and the intent to go one better is also clear:

> Just like real creatures, some agency will act as pets and others will be more like free agents. Some agents will belong to a user, will be maintained by a user, and will live mostly in that user's computer. Others will be free agents that don't really belong to anyone. And just like real creatures, the agents will be born, die and reproduce ... I'm convinced that we need (these agents) because the digital world is too overwhelming for people to deal with, no matter how good the interfaces we design ... Basically, we're trying to change the nature of human computer interaction ... users will not only be manipulating things personally, but will also manage some agents that work on their behalf. (Maes 1995, cited in Suchman 2001, 9)

So genetic algorithms beget something that is meant to approximate life. Whether, as Suchman (2001) convincingly argues, these icons are often based on faulty attributions of interactive agency to machines, the fact remains that these faulty attributions are now constitutive. By enabling and amplifying non-verbal communication between humans, between humans and machines and ultimately between machines and machines, these icons seek to greatly enhance the usefulness of embedded systems.

Software writing space

Wherever we go, then, in modern urbanized spaces, we are directed by software: driving in the car, stopping at the red light, crossing the road, getting into an elevator, using the washing machine or the dishwasher or the microwave, making a phone call, writing a letter, playing a CD or a computer game, the list goes on and on. Given that we have established the prevalence of mechanical writing in the spaces of everyday life, we now need to begin to establish exactly how that effectivity comes about. In this part of the paper we will argue that this effectivity stems from three different but intersecting geographies. The first of these geographies is the most obvious, the large and complex geography of the writing of software – of the production of lines of code – a geography that takes in many different locations and many different languages and which has been built up progressively since the invention of programming in the 1940s. And this geography of writing can only grow, especially given the demands of a modern economy that is built on software (see

Chandler and Cortada 2000) and the consequent need to produce more and more lines of code – whether we are talking of the 3000 or so lines of code that an electric toothbrush may now use, or the millions of lines of code that inhabit a personal computer (Lohr 2001). 'Business cycles and Wall Street enthusiasms will come and go, but someone will have to build all the needed software. Programmers are the artisans, craftsmen, brick layers, and architects of the Information Age' (Lohr 2001, 7).

Software programming is still itself little understood. As we have seen already, it is a curious blend of science, engineering and artistry. On the whole, studies of the programming labour process (e.g. Downey 1998; Orr 1996; Perlow 1997; O'Riain 2000), do not capture its exact blend of writing skills and are unable to explain why some people are quite clearly better at programming than others. As Knuth (1997) has observed:

> There are a certain percentage of undergraduates – perhaps two per cent or so – who have the mental abilities that make them good at computer programming. They are good at it, and its just flows out of them. . . . The two per cent are the only ones who are really going to make these machines do amazing things. I wish it weren't so, but that is the way it has always been. (cited in Lohr 2001, 9)

Whether new developments like grid programming can overcome the reliance on those few who can manipulate the few basic higher-level languages (like C and its variants), or whether large amounts of programming will end up being automated – software produced by software – remains to be seen. Certainly, writing code is

> a curious blend of science, engineering, artistry – and error: even experienced programmers make on average one error for every ten lines of code (and all it takes is three or four defects per 1000 lines of code for software to start doing unpredictable things). (*The Economist* 2002)

Though software programming has a long history spun out of many locations, its geography has now stabilized. To begin with, there are, of course, all these institutions that teach programmers to write code: universities, schools and colleges, computer institutes, and so on. But the geography of programming is most often linked to centres of software production. This geography can be apprehended in two ways. One is the large and complicated geography of those humans who *write* software.

> The globalisation of the information technology industry is seen to result not in a virtual economy but in a global industry organized around and through certain key plans and regions. Within these global workspaces, relations among workers constantly cycle through phases of cohesion and fragmentation, as worker solidarity is mobilised for principles of innovation but disowned by the structure of careers in the labour market. (O'Riain 2000, 179)

The geography of software production is concentrated into a very few key places and regions: Silicon Valley (Kenney 2000), New York (Pratt 2000), London, and a number of subsidiary and sometimes mass production software locations (often concentrating on tasks like consulting, testing and support) in countries like Ireland and India (O'Riain 2000). However, because writing software requires skills that are still in short supply, especially for newer languages, software writers from all over the world often gravitate to the main software-writing centres (Kenney 2000).

The other is a much wider network of production that takes in many consumers as well. Since the founding of the free software movement in 1983 by Richard Stallman, Open Source software has become a massive collective project (McKelvey 2000). This has been particularly the case since the founding of Internet Languages like Linux and Perl, essentially software that has been communally enhanced by thousands of software writers worldwide (Moody 2000). The much trumpeted Linux – with its familiar Penguin logo – was born from three sources: a team of professional corporate developers attempting to solve their organization's needs, commercial software companies and individual programmers. More interesting in many ways is Perl (Practical Evaluation and Report Language). Created in 1987 in Santa Monica by Larry Wall, Perl is now known as the 'duct tape of the internet'. Made visible by the familiar logo of the camel, signifying an entity often thought to have been made by a committee – but which still works – Perl is a kind of mechanical writing culture with something like a million users (Moody 2000). Perl does not operate according to the strict logics necessary to write earlier generation programming languages. Rather, Wall, a linguistics expert, created it to mimic 'expressive' written languages on the principle of 'there's more than one way to do it'. Perl is, in other words, a

language that allows a large amount of creative expression, on the principle that 'easy things should be easy and hard things should be possible' (Wall *et al*. 2000, 4). Thus Perl is not a minimalistic computer language, but has the capacity to be fuzzy and to migrate. However,

> Wall's achievement goes much further than simply a great language. Perl was one of the key enabling technologies that made possible the last wave of interactive web sites, and hence e-commerce itself. It established yet another field – that of programming languages – where the net-based open-development process had produced better results than those traditionally employed within software companies to create their black box products. (Moody 2000, 134)

And, at least in part, this was because of the openness of the development process:

> if you have a problem that the Perl community can't fix, you have the ultimate backstop: the source code itself. The Perl community is not in the business of renting you their trade secrets in the guise of upgrades. (Wall *et al*. 2000, xvii)

Whatever the language, it is clear that open software is being written in many locations at once but, even here, there is a very definite hierarchy of places and people.[11]

The second geography is one of power. Power is built into software from its inception. For example, binary code is premised on the Leibnizian conception of permitted/not permitted. But, that said, software has no fixed foundations beyond the logical and material technologies of micro-informational processing, so we must find a body of work that can play to this characteristic. This we do through the Foucauldian notion of governmentality. Foucault's chief concern was with an analysis of government that took as its central concern *how* we govern and are governed within specific regimes, and the conditions under which such regimes are able to emerge, continue to persist, and are transformed. According to Dean, an analytics of government therefore works through four dimensions of problematization:

> 1 Characteristic forms of visibility, ways of seeing and perceiving
> 2 Distinctive ways of thinking and questioning, relying on definite vocabularies and procedures for the production of thoughts (e.g. those derived from the social, human and behavioural sciences)
> 3 Specific ways of acting, intervening and directing, made up of particular types of practical rationality ('expertise' and 'know-how'), and relying upon definite mechanisms, techniques and technologies
> 4 Characteristic ways of forming subjects, selves, persons, actors or agents. (Dean 1999, 23)

We can see straight away that software intervenes in each of these dimensions. It has changed characteristic forms of visibility by informationalizing space, so producing new objects of analysis. It has changed ways of thinking and questioning by producing new analytic procedures. It has changed the nature of expertise by producing new templates for decisionmaking and it is changing the nature of human subjects by producing enhanced capabilities and by questioning not just what techniques of the self consist of, but whether the self is a meaningful governmental category.

Software, is now, therefore, a key technology of government for both the state and commerce. But it is more than just a potent juridical intermediary. Increasingly, software is becoming the practice of government. What were corporeal routines that could be questioned have seemingly become incorporeal routines that lie below the level of explicit discourse, that are no longer disclosed.

Seen in this way, what then does software consist of, outside the physical fact of lines of code? In essence, we can say that it consists of rules of conduct able to be applied to determinate situations. But these rules of conduct operate at a distance, so that too often the code seems to have little to do with the situations in which it is applied.

> Instead of seeing the program as a real instrument affecting the lives of real people, programmers sometimes see it as a game, a challenge to their ingenuity. The alienating quality of the computer permits us to overlook the human consequences of a programming design or error. An assignment to link scattered databases in different quarters, for example, becomes nothing more than a problem to be solved; the programmer does not notice the resulting diminution of privacy and the increased opportunities for mischief that can result. (Kohanski 1998, 22)

Again, programmers themselves argue about exactly what it is that they are producing and who makes the decisions about the decisions incorporated into the code (Ullman 1997; Kohanski 1998). The remarkably few ethnographies of the labour process in the software industry (e.g. Orr 1996; Perlow 1997; Downey 1998; O'Riain 2000) hardly mention the issue, even though it is surely crucial.

What we can say is that code is law of a kind. But it is not so much law considered as a set of rules as law considered as a set of possible stories framing encounters (Lessig 1999), new *adaptive* standards of conduct. Taken together, these stories write out standards of conduct that can work in all kinds of situations at all kinds of scales. These stories can be simple blocks. They can be encryptions. They can be overall architectures. So, code has different effects on conduct from the given of the green, amber, red of traffic lights to the fact that only 23 people can be present in an AOL chat-room at once (originally a choice of code engineers), to the amount of information able to be collected on each person and translated to commercial advantage. Within certain highly coded domains, this information can be wielded in almost absolute terms:

> AOL space [is] different from other places in cyberspace – it is easier for AOL to identify who you are, and harder for individuals to find out who you are; easier for AOL to speak to all its 'citizens' as it wishes, and harder for dissidents to organise against AOL's views about how things ought to be; easier for AOL to market, and harder for individuals to hide. AOL is a different narrative world; it can create other different worlds because it is in control of the architecture of that world. Members in that space face, in a sense, different sets of laws of nature; AOL makes those laws. (Lessig 1999, 70)

In a sense, what software is able to achieve is a standardization and classification of urban situations in ways that were formerly impossible. It forms a new chapter in what Bowker and Star (1999, 31) call the 'categorical saturation' of the modern world:

> Although it is possible to pull out a single classification scheme or standard for reference purposes, in reality none of them stand alone. So a subproperty of ubiquity is interdependence, and frequently, integration. A systems approach might see the proliferation of both standards and classifications as purely a matter of integration – almost like a giant web of interoperability. Yet the sheer density of these phenomena go beyond questions of interoperability. They are layered, tangled, textured, they interact to form an ecology as well as a flat set of compatibilities. That is to say, they facilitate the collection of heterogeneous 'dispositif techniques' (Foucault 1975). They are lodged in different communities of practice such as laboratories, records, offices, insurance companies) and so forth. There *are* spaces between (unclassified, non-standard areas), of course, and these are especially important to the analysis. It seems that increasingly these spaces are marked as unclassified and non-standard. (Bowker and Star 1999, 31)

Seen in this way, the governmentality of software is perhaps best approximated to by Deleuze's (1990) notion of 'societies of control' in that it provides a set of modulations that constantly direct how citizens act (and who, therefore, is important). Those modulations are often simply opportune. For example, many of the most malign effects of code have arisen when systems are linked in ways that provide new opportunities for surveillance, for example by providing previously unavailable kinds of information. 'As a product manager once told me, ''I've never seen anyone with two systems who didn't want us to hook them together''' (Ullman 1997, 85). But increasingly the modulation will be purposeful. As software becomes increasingly context-aware, so it will be able to adjust rules to circumstances, providing a new kind of mechanical stance to judgement that may well begin to redefine what counts as law.

Numerous examples of new software classification and standardization entities exist. We will fix on one particularly potent one – the humble spreadsheet. Low-cost spreadsheet programs first appeared in 1979 with the introduction of VisiCalc (for Visible Calculator) for the Apple II computer, developed by Daniel Bricklin, Robert Frankston and Dan Fylstra (see Lohr 2001). Within five years, over one million spreadsheets were being sold annually worldwide. Spreadsheets became ever more ubiquitous as a result of additional design work and the ability to be used on a PC (leading to the successor program Lotus 1-2-3 for the IBM PC, and to Microsoft Excel) – as well as more recently to a Linux spreadsheet program. Certainly, within ten years it could already legitimately be claimed that many businesses had become 'spreadsheet cultures'. Subsequently, spreadsheets have continued to evolve with the addition of a WYSIWYG (What You See Is What You Get) interface (with many word processing facilities), as well as sophisticated graphical display capabilities.

The spreadsheet was able to be adapted so rapidly for two reasons. First, because it functioned as a high level programming language that could easily be made task-specific through just a few functions. As Nardi (1995) points out, the spreadsheet was accessible (even in its early completely text-based programming formats) because it used

very simple control constraints, which meant that results could be achieved almost immediately with relatively little effort on behalf of the user. The sacrifice of the flexibility and generality characteristic of most programming languages paid dividends in producing much greater expressivity. Second, the spreadsheet format mimicked the structure of the paper ledger sheet, so that it was already familiar and able to be inserted into the everyday life of business. Yet,

> The spreadsheet is fundamentally different from earlier programs for financial calculation, with their unbridgeable separation between programs and data – responding to a nearly unbridgeable separation between programme and accountant. The key innovation was not in the interface elements or the software structure but rather in the virtuality provided to the user – the underlying world of objects and their behaviour. The spreadsheet virtually combines the regular structure of the familiar ledger sheet with an underlying structure of interlinked formulas. The non-technical user can build a complex financial model incrementally, and can explore the model through successive iterations of inputs. This quantitative change in ease meant a qualitative change in how people worked with data. (Winograd 1996, 230)

Thus managerial behaviour was profoundly changed by the generation of new kinds of information *and* interaction, new 'everyday business lives':

> The artfully crafted spreadsheet – here is how we can cut costs! Here's how we should restructure this deal! – could and did prove as politically explosive a document as the declaration of independence or the communist manifesto. Spreadsheet software breathed new life into the old adage 'figures don't lie, but liars figure'. As budgets and forecasts were used to find previously unimagined opportunities, traditional perceptions of power, politics, productivity, and profit all dramatically shifted. Finance-driven organisations often found themselves reorganising around their spreadsheets. (Schrage 2000, 39)

The spreadsheet therefore had impacts in a number of ways. To begin with, it provided new opportunity for interaction. Then, it furnished managers with rhetorical energy (there is a rhetoric of spreadsheets that results in more or less persuasive forecasts: 'spreadsheet rhetoric is about turning what appears to be a dispassionate logic of numbers into presentations that mute opposition even as they enlist allies' (Schrage 2000, 47). Again, the spreadsheet asked new questions, tested new ideas and provided new business opportunities (such as new financial products). They were 'a medium for serious play as much as for crunching hard numbers' (Schrage 2000, 44). And, best of all, they provided a new language. Thus Schrage exaggerates only slightly when he writes that:

> The multinational, multibillion dollar institutional leader Asea Brown Boveri, nominally headquartered in Europe, insists that English is its primary language, but in practice the dominant language of ABB is spreadsheet. Indeed, several senior managers observe that it is communication, disagreement and negotiation over spreadsheet forecasts and projects that drive business at ABB. 'Yes, I think it is better to be more fluent in your spreadsheet forecasts than in English', says a senior ABB executive. 'Our numbers are probably more important to us than our words'. (Schrage 2000, 46–7)

In other words, spreadsheets created new stories of government, not least by producing new intellectual stimulation. The artefact of spreadsheet program created new coalitions, new forces, new realities. In turn, their persuasiveness as a model allowed them to migrate outwards from business and finance into the city as a whole. They had found a 'design foothold'. So, for example, specialist spreadsheet programs are now available for applications as diverse as car purchase, class notes and assignments, diving decompression, seed cultivation and trading, landfill gas production, archaeological digs, laboratory management, chemical properties, music production and real estate management.

> Now that the spreadsheet is widely available, it has come to be used for many tasks that have nothing to do with figures. A spreadsheet can be used for anything that calls for calculating regular arrays of values that are interconnected by regular formulas – especially for those actualities that call for exploring alternatives. Professors use spreadsheets for grading courses, scientists use spreadsheets for interpreting data from experiments, and builders use spreadsheets for keeping track of materials. New kinds of spreadsheets have been developed that fill the cells with visual images, sounds, and other data representations, interleaved by formulas that perform calculations in the appropriate domain. (Winograd 1996, 231)

But Foucault's notion of governmentality has a negative side. Though it stresses the positive role of power, it overwhelmingly concentrates on the realm of constraint, whether legislated or self-imposed. But there is a third kind of geography of software, what Hobart and Schiffman (1998) call

'the realm of play'. The general profusion of software, its increasing complexity and consequent emergent properties, all count as means of producing playful idioms that have not been captured by dominant orders. Software's very indeterminacy and lack of closure provide a means of creating new kinds of order.

> Play is 'freedom', wrote Huizinga, yet it also 'creates order, *is* order'. In its creative activity, play does not imitate or reflect or correspond or map directly to an outside world, although the resulting order it produces might eventually do so. Play's own order is not derivatively mimetic but *sui generis*, methectic, as the Greeks would say in reference to the theatre, a 'helping-out-of-the-action'. By now it should be abundantly clear that information in the contemporary idiom means process, always in motion, always abetting the action of life's drama. Less clear, perhaps, this motion has direction, not towards any telos, purpose, or end, but following with its rules the arrow of time. From the time-bound movement of information play there can emerge novel, unforeseen structures, 'order for free'. (Hobart and Schiffman 1998, 259)

The creative phrasing of play can be put another way, as a tending of the virtual. Software has creative potential that is more than just possibility. As Lévy puts it,

> The possible is already fully constituted but exists in a state of limbo. It can be realised without any change occurring either in its determination or nature. It is a phantom reality, something latent. The possible is exactly like the real, the only thing missing being existence. The realisation of a possible is not an act of creation in the fullest sense of the word, for creation implies the innovative production of an idea or form. The difference between the possible and the real is thus purely logical.
>
> The virtual should, properly speaking be compared not to the real but the actual. Unlike the possible, which is static and already constituted, the virtual is a kind of problematic complex, the kind of tendencies or forces that accompanies a situation, event, object, or entity, and which involves a process of resolution: actualisation. This problematic complex belongs to the entity in question and even constitutes one of its primary dimensions. The seed's problem, for example, is the growth of the tree. The seed *is* this problem, even if it is also something more than that. This does not signify that the seed knows exactly what the shape of the tree will be, which will one day burst into bloom and spread its leaves above it. Based on its internal limitations, the seed will have to invent the tree, co-produce it together with the circumstances it encounters. (Lévy 1998, 24)

Seen in this way, software does not have to be seen as simply a form of constant and unbending recital. It can be redefined as an experimental tool, such that

> rather than being defined principally through its actuality (a solution), the entity now finds its essential consistency within a problematic field. The virtualisation of a given entity consists in determining the general question to which it responds, in mutating the entity in the direction of this question and redefining the initial actuality as the response to a specific question. (Lévy 1998, 26)

Given solution, then, proceeds to a different problem. So software that may be designed to produce defined and determined responses can in certain circumstances – as the example of spreadsheets shows – act as a spur to forms of creativity that can transgress these standard forms of classificatory arrangement: there are new forms of Yes as well as No (Lunenfeld 1999). Software is, then, being used in remarkably creative and craft-laden ways – from advances in animation that are redefining kinetic events (Wells 1998), through work on new forms of music using musical instrument digital interfaces (MIDI), through work on new forms of theatre and dance (Sparacino *et al.* 2000), through the pairing of artists and technologists (Harris 1999), through to the work of technologically sophisticated artists like Char Davies, who use software to create new virtual art forms which recognize that 'whenever people experience a piece of software – whether it be a spreadsheet or a physics simulation [they can] experience beauty, satisfaction and fun or the corresponding opposites' (Winograd 1996, xix). Such aesthetic uses all depend on software programs, from the early (mid-1980s) Macintosh programs like MacPaint and MacDraw to complex contemporary multimedia programs like Director. Such software can be said to proceed in a number of different but related ways. To begin with, it can be used to extend the range and meaning of the body, not least by conjuring up various affective states (McCullough 1996). It is interesting that so many creative software projects have been concerned with the body, both representing it in new ways, and questioning its bounds as it is constantly augmented (e.g. Allsopp and de Lahunta 1999). Indeed, new software developments like haptic computing promise to extend the range of informationalized embodiment still further, producing

interfaces between software and body which will allow effects of unparalleled subtlety to be created. As Tenhaaf notes:

> the body seems . . . to know itself in a minutely expressive way that may by definition not be conscious, but that may nevertheless become more available [as] up-to-the-minute information and imagery about the body's deepest workings and its most complicated biosocial functions . . . now enter and become interwoven into the biomedical readability of the self. (Tenhaaf 1996, 67–8)

Then, software can be used to question familiar Euro-American notions of representation. After all, in certain senses

> we are moving into an information environment in which there is to be no representation at all. How we represent ourselves and form subjectivity, in relation to the real world, will no longer be an issue, as simulated entry into parallel digital worlds comes to supercede this relationship. (Tenhaaf 1996, 52; see also Brooks 1991)

Too strong, perhaps, yet it is not difficult to see how many artists – and programmers – are using software to question conventional notions of representation in ways that are very close to Strathern's recasting of Euro-American perspectivism.

> Suppose, instead of a Renaissance imagination which at times tried to make the whole world the singular object of the viewer's vision, having a perspective were regarded as a capacity belonging to animate life. What the writer could 'see' would be other life forms. What would be finite here? Could it be the manner in which one's perspective was returned to one? That is, closure would lie in the fact that one simultaneously had one's own perspective and received the perspective of another. Or, rather, the point at which that viewer was conscious that he or she had a perspective on things would be the point at which he or she would meet (so to speak) the reciprocal perspectives of other life forms. (Strathern 1999, 249)

Again, software, precisely because it is an imperfect medium, can produce all kinds of opportunities for what Deleuze (1994) called 'bad mimesis'. Many forms of software are procedures that contain 'flaws' that can be used to produce new and interesting potentialities: the culture of the not-quite copy (Schwartz 1997). And, finally, it is worth remembering how much software innovation has itself found its momentum from a line of dissident programmers (Moody 2000; Lohr 2001) in the tradition of Thomas Nelson and Douglas Engelbart, who not only stand in line as part-inventors of hypertext, the World Wide Web, and so on, but were clearly concerned with using software to boost human capacities, and equally concerned with redefining what was 'human'.

The rise of semi-artificial life

Our problem in thinking about the automatic production of space by, with and from software is, by now, we hope, clear. That is, that we are schooled to ignore this kind of mechanical writing. It is a part of the taken-for-granted background, to the extent that some have suggested that it constitutes a kind of virtual skin (or set of skins) around human bodies (Bailey 1996; Lévy 1998), one that will become an ever better fit as wearable systems become common.

So how can we summarize software's effectivity as a new means of populating space, and as a new means of reproducing intelligence? As we have written this paper, so we have vacillated between two answers, exactly as the literature does. One answer is to see software as an epochal event. Software signals a fundamental reorganization of the environment, a vast system of distributed cognition through which the environment increasingly thinks for itself, an extra layer of thinking.[12]

Seen in this way, software is a part of the extended organism of a new form of humanity, a kind of extended phenotype in which the environment we have made speaks back to us: 'organisms are integral with the world outside them' (Turner 2000, 6). If it is the case that modern biology of the kind typified by developmental systems theory is no longer sure what organisms are, so we can no longer be sure what humanity consists of – where it stops and something else begins. In this sense, software is building using writing. Rather than bricks and concrete, though, we have words and strings of words. And, of course, increasingly buildings may well become as much words as bricks. Mitchell may exaggerate the possibilities (see Bolter and Grusin 1999), but even so we can see that some

> buildings of the near future will function more and more like large computers, with multiple processors, distributed memory, various devices to control, and network connections to take care of. They will suck in information from their interiors and surroundings, and they will construct and maintain complex, dynamic

> information overlays delivered through inanimate devices worn or held by inhabitants, screens and spaces in the walls and ceilings, and projections onto enclosing surfaces. The software to manage all this will be a crucial design concern. The operating system for your house will become as essential as the roof, and certainly far more important than the operating system for your desktop PC.
>
> Consequently a growing proportion of a building's construction cost will go into high-value factory-made electronics-loaded software – programmed computers and subsystems, a correspondingly decreasing proportion will go into on-site construction of the structure and cladding. As they become denser with wiring and electronic devices buildings will become more like large-scale printed circuit boards than dumb wallboard. (Mitchell 1999, 65)

What we see – and will see more of – in Euro-American societies, therefore, is a complex ecology of software, 'wideware' (Clark 2000), which will increasingly inhabit every nook and cranny of human life. Software as a kind of thinking grass. Latour and Hernant's (1997) Paris, made up of a vast panorama of signs, directions and objects will be paralleled in the virtual realm, producing a more effective (because polycentric, multi-vocal and heterogeneous) assemblage of control, whole species of normativity interacting in as yet-to-be determined ways. But maybe this answer is a bit too grand and we need to turn to another answer instead, one that is less sweeping, more mundane. After all, to begin with, there is plenty of evidence that software does not always work that well (for example, in the case of surveillance cameras, see Lianos and Douglas 2000) and that much of the writing that takes place in the world will continue to be paper-based, not electronically coded (Sellen and Harper 2002). Then, most software is still initially written by human hands (McCullough 1996). It arises from the striking of keyboards, the clicking of the mouse, the shifting of notes around on a desktop and the consulting of manuals to find various standards and classifications (Bowker and Star 1999, 39). Software still flows from the interface between body and object – even though that interface may be what Knorr-Cetina calls 'post-social', resting on the fact that

> individuals in some areas relate to 'some' objects not only as 'doers' and 'accomplishers' of things within an agency framework but as experiencing, feeling, reflexive and remembering beings – as bearers of the sort of experiences we tend to reserve for the sphere of inter-subjective relationships. (Knorr-Cetina 2001, 522)

And, finally, software is often written with what can only be described as 'human' concerns in mind. In particular, it is one of the key moments of 'virtualism' (Carrier and Miller 1998), the proliferation of theories about the world that prompt efforts to make the world conform, rather than vice versa. Examples of such virtual moments abound in code, from the mobilization of biological analogies discussed above, through various exchanges with complexity theory (Thrift 1999), through numerous philosophical theories – from Heidegger (Winograd and Flores 1987) to Bakhtin (Sparacino *et al*. 2000) – to the use of Kevin Lynch's (1966) model of the city as a means to imagine the internal coherence of software (Dieburger and Frank 1998). And 'human' concerns echo through software in other ways too. So the production of software is increasingly bound up with and incorporates an ethnographic mode of enquiry, whether that be work on more social interfaces, on CSCW, or on the psychology of artificial social actors (Dryer *et al*. 1999). In other words, an ethnographic model of human encounters defines the horizon of certain forms of software.

So, maybe instead of understanding software as writing the next chapter in the evolution of humanity (Leroi-Gourhan 1993; Lévy 1998), we can see it as a more practical extension of human spaces, consisting of three different processes. The first is a simple extension of textuality. So, for example, modern Western cities are effectively intertextual – from the myriad forms issued by bureaucracies, through the book, the newspaper and the web page, through the checkout till roll and the credit card slip to the letter, the e-mail and the text message, the city is one vast intertext. Cities are quite literally written and software is the latest expression of this cursive passion. Second, software is a part of the paraphernalia of everyday urban life revealed by the turn to the noncognitive. It is one of those little but large technologies that are crucial to the bonding of urban time and space, technologies like the pencil (Petroski 1992) and the screw (Rybczynski 2000), which in their very ubiquity go largely unnoticed. We can think of software in this way – as a holding together accomplished through the medium of performative writing. Third, we can see software as a means of transport, as an intermediary passing

information from one place to another so efficiently that the journey appears effortless, movement without friction (Latour 1997). As a running description of the present.

Whatever the answer (and our inclination is to choose both at once), it seems certain that we can no longer think of city spaces in the old, time-honoured ways. Software challenges us to think 'beyond living', 'to mime an absent origin, ''life''' (Doyle 1997, 132). Software challenges us to re-inscribe what we comprehend as inscription. And, most importantly, software challenges us to understand new forms of technological politics and new practices of political invention, legibility and intervention that we are only just beginning to comprehend as political at all: politics of standards, classifications, metrics, and readings (Barry 2001). These orderings – written down as software – are becoming one of the chief ways of animating space. They should not be allowed to take us unaware. One of the more pressing contemporary political tasks must therefore be to design friendlier 'information ecologies' (Nardi and O'Day 1999)[13] that, because of their diversity of outcome, will allow us to shape overlapping spatial mosaics in which effective participation is still possible and still necessary. Automatic can be for the people.

Acknowledgements

We would like to thank audiences at the Royal College of Art, the Conference on Information and the Urban Future held at New York University, and the Conferences on Hegemonies and Code held at the University of Lancaster for their helpful comments on this paper. In particular, we would like to thank Fred Botting, Howard Caygill, Robert Cooper, Mick Dillon, Paul Fletcher, Steve Graham, John Hughes, Patrick Keiller, James Griesemer, Kirsten McAllister, Paolo Palladino, Lucy Suchman and Wes Sharrock for interventions that have much improved the logic and content of this paper. The assistance of the National University of Singapore in providing the resources to allow the paper to be finalized is gratefully acknowledged.

Notes

1 The first published use of the term 'software' as a computing term dates only from 1958, when it was used by John Tulley in the *American Mathematical Monthly* (see Shapiro 2000; Lohr 2001). The *Oxford English Dictionary* now defines software as the 'programs and procedures required to enable a computer to perform a specific task, as opposed to the physical components of the system'. In simple terms, the function of such programs is to process data. Through the employment of various mathematical algorithms, programs process data to produce a requisite output, thus enabling systems to operate. Again in simple terms, software development involves the writing of programs by software developers and computer programmers drawing upon various languages and programming techniques. The language and the programming techniques employed will depend upon the task in question.

2 This convergence begins with inventions like the Jacquard Loom, and has a very definite history of its own, which we have no space to document here. But see Hobart and Schiffman (1998).

3 The following Y2K audit was conducted through the internet; in particular, see the Websites listed in Table I.

4 There are obvious dangers here of information overload, with consumers simply switching off, which have led to a number of attempts by the advertising industry to produce protocols that will allow selection. Taken from http://coverage.cnet.com/Content/Gadgets/Special/FunToWear/ss03html

5 Of course, it is possible to overstate the case mightily. One of us was told of how MIT Media Lab brought together a number of major kitchen manufacturers to consider how kitchen devices could communicate with each other, only to agree that there would not be a lot for them to say.

6 The Audit Commission (1998, 5) defines embedded systems as 'devices used to control, monitor or assist the operation of equipment, machinery or plant . . . [a]ll embedded systems are, or include, computers'.

7 Evolutionary computing attempts to 'breed' progressively better solutions to complex problems, in the same way that successive generations of life forms evolve to cope with particular environments.

8 Of the three terms, soft computing appears to be the broadest, with Bonissone (1998) identifying elements from evolutionary computing (genetic algorithms), fuzz computation (fuzzy logic) as well as probabilistic reasoning (including chaos theory and parts of learning theory (Zadeh 1994)) and neural networks within this category.

9 By which is meant problems with too many possible solutions to try them all within a reasonable period of time (http://ai.about.com/compute/ai/library/weekly/aa110800a.htm).

10 The genetic operator utilizes theories of evolution such as gene crossover and gene mutation to alter the composition of 'children' during reproduction.

11 Not too much should be made of the perception of languages like Linux as being the project of offbeat

code composers. Linux's core has mainly been developed by seven people – Linus Torvalds and David Miller in California, David Cox in England, Ingo Molnar in Hungary, Steven Tweedy, and Ted T'so on the west coast of America and Andrea Arkangel in Europe, who filter the patches of code they receive and pass them on to 250 programmers worldwide, the vast majority of whom work on testing (Young 1999).

12 We often turn, in this regard, to a lecture one of us attended by John Seely Brown at UC Berkeley in May 2001 in which he outlined a project in which every tree in a forest would have a chip inserted into it, producing a natural system with memory and some communication abilities (for example, being able to call for help). An initial sceptical reaction was followed by the thought that this may well indeed be how spaces with software will come to be regarded, as a second informationalized nature (Seely Brown and Duguid 2000).

13 Nardi and O'Day, for example, point to the role of 'gardeners', mediators with computer skills above the average who are found in many walks of life and act as keystones in communities, and as the main innovators in the co-evolution of tools and practices. Similarly, we need new simpler kinds of 'block' programming languages (rather like MIT Media Lab's Logo), which are task-specific and accessible to all.

References

Aarseth E J 1997 *Cybertext. Perspectives on ergodic literature* Johns Hopkins University Press, Baltimore

Agre P 1997 *Computation and human experience* Cambridge University Press, Cambridge

Allsopp S and de Lahunta S 1999 'On line' Special issue of *Performance Research* 4 20–42

Anderson J A and Rosenfeld E eds 1999 *Talking nets. An oral history of neural networks* MIT Press, Cambridge, MA

Appelbaum D 1995 *The stop* State University of New York Press, Albany

Audit Commission 1998 *A stitch in time: facing the challenge of the year 2000 date change* The Audit Commission, London

Baber C, Haniff D J and Woolley S I 1999 Contrasting paradigms for the development of wearable computers *IBM Systems Journal* 38 18–23

Bailey C 1996 Virtual skin: articulating race in cyberspace in **Moser M A and McLeod D** eds *Immersed in technology. Art in virtual environments* MIT Press, Cambridge, MA 29–50

Barry A 2001 *Technological politics* Athlone Press, London

Bass T 1997 *The predictors* Harper, New York

Belew R K and Vose M D 1997 *Foundations of genetic algorithms 4* Morgan Kaufmann, San Francisco

Bentley P 2001 *Digital biology* Headline, London

Bergman E ed 2000 *Information appliances and beyond* Academic Press, San Diego

Billinghurst nd Quoted in 'Head to toe' CNET.com (http://coverage.cnet.com/Content/Gadgets/Special/FunToWear/ss03.html) Accessed 29 December 2000

Bolter J D and Grusin R 1999 *Remediation. Understanding new media* MIT Press, Cambridge, MA

Bonissone P P 1998 Soft computing and hybrid systems in **Ruspini E, Bonissone P P and Pedrycz W** eds *The handbook of fuzzy computation* Institute of Physics Publishing, London 161–78

Borgmann A 1999 *Holding on to reality. The nature of information at the turn of the millennium* University of Chicago Press, Chicago

— 2000 Semi-artificial life in **Wrathall M and Malpas J** eds *Heidegger, coping and cognitive science. Essays in honor of Herbert L Dreyfus Volume 2* MIT Press, Cambridge, MA 197–205

Bowker G C and Star S L 1999 *Sorting things out. Classification and its consequences* MIT Press, Cambridge, MA

Brooks F 1995 *The mythical man-month* Addison Wesley, San Francisco

Brooks R A 1991 Intelligence without representation *Artificial Intelligence Journal* 47 139–59

Brown B, Green N and Harper R eds 2002 *Wireless world. Social and interactional aspects of the mobile age* Springer Verlag, London

Brown P *et al.* 2000 Context awareness: some compelling applications (www.cks.ex.ac.uk/~pjbrown/papers/acm.html) Accessed 26 April 2001

Carrier J and Miller D eds 1998 *Virtualism. A new political economy* Berg, Oxford

Cassell J, Sullivan J, Prevost S and Churchill E eds 2000 *Embodied conversational agents* MIT Press, Cambridge, MA

Chandler A D and Cortada J W eds 2000 *A nation transformed by information* Oxford University Press, New York

Clark A 2001 *Mindware. An introduction to the philosophy of cognitive science* Oxford University Press, Oxford

Clough P T 2000 *Autoaffection. Unconscious thought in the age of technology* University of Minnesota Press, Minneapolis

Collins H and Kusch M 1998 *The shape of actions. What machines can do* MIT Press, Cambridge, MA

Crystal D 2001 *Language and the internet* Cambridge University Press, Cambridge

Davis L and Steenstrup M 1990 Genetic algorithms and simulated annealing: an overview in **Davis L** ed *Genetic algorithms and simulated annealing* Pitman, London 46–52

Dean M 1999 *Governmentality. Power and rule in modern society* Sage, London

Deleuze G 1990 Postscript to societies of control *October* 31 27–36

— 1994 *Difference and repetition* Columbia University Press, New York

Deleuze G and Parnet C 1987 *Dialogues* Athlone Press, London

Derrida J 1998 *Archive fever* Chicago University Press, Chicago

Dieburger A and Frank A U 1998 A city metaphor for supporting navigation in complex information spaces *Journal of Visual Languages and Computing* 9 597–622

Downey G C 1998 *The machine in me. An anthropologist sitting among computer engineers* Routledge, New York

Downey J and McGuigan J eds 1999 *Technocities* Sage, London

Doyle R M 1997 *On beyond living. Rhetorical transformations of the life sciences* Stanford University Press, Stanford

Dreyfus H 2001 *On the internet* Routledge, New York

Dryer D C, Eisbach C and Ark W S 1999 At what cost pervasive? A social computing view of mobile computing systems *IBM Systems Journal* 38 61–5

Economist 2002 A lemon law for software *The Economist* 16 March Technology Quarterly, 3

Economist Intelligence Unit 2000 *Electronic revolution in the motor industry* The Economist, London

Fischer S R 2001 *A history of writing* Reaktion, London

Foucault M 1975 *The order of things* Tavistock, London

Gleick J 1999 *Faster. The acceleration of just about everything* Little Brown, London

Goldberg D E 1989 *Genetic algorithms in search, optimization and machine learning* Addison-Wesley, London

Graham S 1998 Spaces of surveillant simulation: new technologies, digital representations, and material geographies *Environment and Planning D Society and Space* 16 483–504

Hansen M 2000 *Embodying technesis. Technology beyond writing* University of Michigan Press, Ann Arbor

Harris C ed 1999 *Art and innovation. The xerox parc. Artist-in-residence program* MIT Press, Cambridge, MA

Hayles N K 1999 *How we became posthuman* MIT Press, Cambridge, MA

Heitkötter J and Beasley D 1999 The hitch-hiker's guide to evolutionary computation (http://www.cs.bham.ac.uk/Mirrors/ftp.de.uu.net/EC/clife/www/top.htm) Accessed 18 April 2001

Helmreich S 1998 *Silicon second nature: culturing artificial life in a digital world* University of California Press, Berkeley, CA

Hobart M E and Schiffman Z S 1998 *Information ages. Literacy, numeracy and the computer revolution* Johns Hopkins University Press, Baltimore

Holland J H 1975 *Adaptation in natural and artificial systems* MIT Press, Cambridge, MA

Horvath R J 1974 Machine space *The Geographical Review* LXIV 166–87

Johnson C 1993 *System and writing in the philosophy of Jacques Derrida* Cambridge University Press, Cambridge

— 1999 Ambient technologies, meaning signs *Oxford Literary Review* 21 117–34

Johnson S 1997 *Interface culture* Basic Books, New York

Kay L 2000 *Who wrote the book of life? A history of the genetic code* Stanford University Press, Stanford, CA

Kenney M ed 2000 *Understanding Silicon Valley. The anatomy of an entrepreneurial region* Stanford University Press, Stanford, CA

Khiang Tam K, Khalid M and Yusuf R 1996 Intelligent traffic lights control by fuzzy logic *Malaysian Journal of Computer Science* 9 66–71

Kinnes S 2000 In your face *The Guardian* 10 February 14–15

Kittler F A 1997 *Literature, media information systems* OPA, Amsterdam

Knath D 1997 *The art of computer programming* (two volumes) Addison-Wesley, San Francisco

Knorr-Cetina K 2001 *Handbook of social theory* Sage, London

Kohanski D 1998 *The philosophical programmer. Reflections on the moth in the machine* St Martins Press, New York

Latour B 1997 Trains of thought: Piaget, Formalism and the Fifth Dimension *Common Knowledge* 6 170–91

Latour B and Hernant E 1997 *Paris. Ville invisible* Institut Synthelabo, Paris

Laurel B 1993 *Computers as theatre* Addison Wesley, Boston

Laurier E and Philo C 2001 'Accomplishing the company region with a car, mobile phone, cardboard cut-out, some carbon paper and a few boxes' Paper presented at the Annual Meeting of the Association of American Geographers, February–March 2001

Lechte J 1999 The who and what of writing in the electronic age *Oxford Literary Review* 21 135–60

Leroi-Gourhan A 1993 *Gesture and speech* MIT Press, Cambridge, MA

Lessig L 1999 *Code and other laws of cyberspace* Basic Books, New York

Lévy P 1998 *Becoming virtual. Reality in the digital age* Plenum, New York

Lohr J 2001 *Go to. Software superheros from Fortran to the internet age* Basic Books, New York

Lunenfeld P ed 1999 *The digital dialectic. New essays on new media* MIT Press, Cambridge, MA

Lynch K 1966 *The image of the city* MIT Press, Cambridge, MA

Mackenzie D 1996 *Knowing machines. Essays on technical change* MIT Press, Cambridge, MA

McCarthy A 2001 Ambient television Duke University Press, Durham, NC

McCullough M 1998 *A digital craft. The practised digital hand* MIT Press, Cambridge, MA

McKelvey M 2000 The economic dynamics of software: comparing Microsoft, Netscape and Linux *Industrial and Corporate Change* 11 23–42

Mitchell W T J 1999 *E-Topia* MIT Press, Cambridge, MA

Moody G 2000 *Rebel code* Allen Lane, London

Nardi B A 1995 *A small matter of programming. Perspectives on end user computing* MIT Press, Cambridge, MA

— ed 1996 *Context and consciousness. Activity theory and human–computer interaction* MIT Press, Cambridge, MA

Nardi B A and O'Day V L 1999 *Information ecologies. Using technology with heart* MIT Press, Cambridge, MA

New Scientist 1999a Caught on camera 25 September Reed Business Information, London

— 1999b Warning! Strange behaviour 11 December Reed Business Information, London

— 2000 Emerging technologies: everything, anywhere 21 October Reed Business Information, London

— 2001 Write here, write now *New Scientist* 1 December 2001 16–17

Norman D A 1998 *The invisible computer* MIT Press, Cambridge, MA

— 2000 Making technology visible: a conversation with Don Norman in **Bergman D** ed *Information appliances and beyond. Interaction design for consumer products* Morgan Kaufman, San Francisco 9–26

Omojola I *et al.* 2000 An installation of interactive furniture *IBM Systems Journal* 39 82–90

O'Riain S 2000 Net-working for a living: Irish software developers in the global workplace in **Burawoy M, Blum J and Sheba G** eds *Global ethnography. Forces, connections and imaginations in a postmodern world* University of California Press, Berkeley 175–202

Orr J E 1996 *Talking about machines. An ethnography of a modern job* Cornell University Press, Ithaca

Oyama S 2001 *The ontogeny of information* 2nd edn Princeton University Press, Princeton

Paradiso J A, Hsiao K, Benbasat A Y and Teegarden Z 2000 Design and implementation of expressive footwear *IBM Systems Journal* 30 103–15

Perlow L A 1997 *Finding time* Cornell University Press, Ithaca

Pesce M 2000 *The playful world: how technology is transforming our imagination* Ballatine Books, New York

Petroski H 1992 *The pencil* Knopf, New York

Phelan P 1993 *Unmarked* Routledge, New York

Picard R W 1997 *Affective computing* MIT Press, London

— 2000 Toward computers that recognise and respond to use emotion *IBM Systems Journal* 39 16

Plutowski M 2000 Emotional computing (www.emotivate.com/Book/intro.htm) Accessed 1 February 2001

Post E R, Orth M, Rosso P R and Gershenfeld N 2000 E-broidery: design and fabrication of textile-based computing *IBM Systems Journal* 39 142–53

Poster M 2001 *What's the matter with the internet?* University of Minnesota Press, Minneapolis

Pratt A 2000 New media, the new economy and new spaces *Geoforum* 31 425–36

Ruspini E H, Bonissone P P and Pedrycz W 1998 eds *Handbook of fuzzy computation* Institute of Physics Publishing, Bristol

Rybczynski W 2000 *One good turn. A natural history of the screwdriver and the screw* Simon and Schuster, New York

Schrage M 2000 *Serious play* Harvard Business School Press, Boston

Schwartz H 1997 *The culture of the copy* Zone Books, New York

Seely Brown J and Duguid P 2000 *The social life of information* Harvard Business School Press, Boston

Sellen A J and Harper R H R 2002 *The myth of the paperless office* MIT Press, Cambridge, MA

Shapiro F 2000 Origins of the term software: evidence from the JSTOR electronic journal archive *IEEE Annals of the History of Computing* 22 69–71

Sipper M 2000 A brief introduction to genetic algorithms (http://lslwww.epfl.ch/[]moshes/ga_main.html) Accessed 17 April 2001

Sparacino F, Davenport G and Pentland A 2000 Media in performance: interactive spaces for dance, theatre, circus and museum exhibits *IBM Systems Journal* 39 160–73

Spivak G C 1993 *Outside in the teaching machine* Routledge, New York

Stiegler B 1998 *Technics and time 1: The fault of Epimetheus* Stanford University Press, Stanford

Strathern M 1999 *Property, substance and effect* Athlone Press, London

Suchman L 1987 *Plans and situated actions. The problem of human machine communications* Cambridge University Press, Cambridge

— 2001 Human/machine reconsidered Department of Sociology, University of Lancaster (http://www.comp.lancs.ac.uk/sociology/soc04015.htm/) Accessed 26 January 2002

Tenhaaf N 1996 Mysteries of the bioapparatus in **Moser M A and McLeod D** eds *Immersed in technology. Art in virtual environments* MIT Press, Cambridge, MA 51–71

Thomas P J ed 1995 *The social and interactional dimensions of human–computer interfaces* Cambridge University Press, Cambridge

Thrift N J 1999 The place of complexity *Theory, Culture and Society* 16 31–70

— 2000 Afterwords *Environment and Planning D. Society and Space* 18 213–55

— 2001 Elsewhere in **Cummings N and Lewandowska M** eds *Capital* Tate Modern, London

Townsend A 2000 Life in the real-time city: mobile telephones and urban metabolism *Journal of Urban Technology* 7 85–104

Turkle S 1991 Romantic reactions: paradoxical responses to the computer presence in **Sheehan J J and Sosna M** eds *Boundaries of humanity: humans, animals, machines* University of California Press, Berkeley 63–72

Turner J S 2000 *The extended organism. The physiology of animal-built structures* Harvard University Press, Cambridge, MA

Ullman E 1997 *Close to the machine. Technophilia and its discontents* City Lights, San Francisco

Ulmer B and Ishii H 2000 Emerging broadcasts for tangible user interfaces *IBM Systems Journal* 30 65–72

Ulmer G 1989 *Teletheory* Routledge, London

Vidler A 2000 *Warped space* MIT Press, Cambridge, MA

Wall L, Christiansen T, Orwant J 2000 *Programming Perl* O'Reilly, Sebastopol, CA

Weiser M, Gold R and Brown J S 1999 The origins of ubiquitous computing research at PARC in the late 1980s *IBM Systems Journal* 38 83–97

Wells P 1998 *Understanding animation* Routledge, London

Winograd T ed 1996 *Bringing design to software* Addison Wesley, Reading, MA

Winograd T and Flores F 1987 *Understanding computers and cognition. A new foundation for design* Addison Wesley, Reading, MA

Young R 1999 *Under the radar. How Red Hat changed the software business* Coriolis, Scottsdale, AZ

Zadeh L A 1994 What is BISC? (BISC website http.cs.berkeley.edu/projects/Bisc/bisc.welcome.html) Accessed 13 April 2001