ISSN: 2042-034X

URBIS RESEARCH FORUM REVIEW

Science and the City

Vol 1, Issue 2

Code/Space:
Martin Dodge

"Software is everything. In the history of human technology, nothing has become as essential as fast as software."
(Fishman 1996: 95)

Introduction

Software is all about us, animating city functions, monitoring infrastructures, regulating flows and enrolled in a myriad of daily activities. As geographers Nigel Thrift and Shaun French (2002: 309) have noted "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." There is a need for novel urban research that documents and accounts for how the socio-technical governance of cities is becoming automatic and character based on the calculative power and anticipatory capacities of computer software. To begin to provide a comprehensive account of how code makes so much of contemporary cities work is a challenge as many people do not comprehend software and much computing is 'disappearing' into what Thrift (2004) has called the 'technological unconscious'. First off research needs to account for the tremendous scale and speed of growth in the extent of code, as suggested by Fishman's decade old statement quoted above, and then also to understand the productive and creative power that software has to make the world differently in terms of the materiality, economic relations and social processes at the heart of city life.

Research into the urban landscapes of code can usefully progress from the position of non-representational theory in which the analytical lens shifts focus from an ontological description (what something is) to ontogenesis (how something comes to be). So the city is no longer read as a set of fixed, geometric spaces and 'hard' objects but instead in the ontogenetic reading spaces are seen to emerge through practice and the city is made up of all manner of provisional objects that only take on form, function and meaning through how they are performed. Software, I would argue, increasingly makes a difference to spatial practices and material performances. Software is enrolled to bring the city into being in particular ways.

In the rest of this article I set out ideas on how we might productively research the ways that software is enrolled into contemporary urban practices, beginning by defining the nature of code, then considering some of the significant theoretical ideas advanced by geographical scholars in the last decade to account for the spatiality of software. Lastly, I seek to exemplify some of these concepts drawing on empirical evidence from Anglo-American cities and focusing on the challenges of urban mobility and car driving.

Defining code

We all work with code consciously when we directly interact with computer devices and more widely in the many unconscious brushes with software when our activities come within the orbit of coded systems (e.g., paying by card in a store depends on layers of software to make the monetary transaction proceed as intended). We may be aware of its effects but most of us have little idea about what software is or how it works.

Software animates a variety of computing machines, some designed as devices for humans to directly relate to, but increasingly the machines that host code are 'black-boxes' installed discretely in the background without the need to address people to do their work.

It can be argued that it is hard to get a solid sense of software because so much code is hidden from view, and even when one can see the computer carapace it is not possible to observe the nature of the code working. Holding a mobile phone for example, one can see a surface presentation generated by the interface code on the screen, but this tells you nothing of what is running 'beneath' the display. We might use a laptop for hours on a daily basis for work and leisure becoming intimately familiar with its materiality, its weight, the noise of the cooling fan, the feel of the keyboard and so on, but how much do we know of the software running on it? As Annette Schindler notes "you think you know your computer, but really all you know is a surface on your screen." (quoted in Mirapaul 2003).

Conceptually software is built of lines of code – simple instructions and algorithmic rules – that when combined together with appropriate data produce operative programs capable of complex functions. Software could be thought of as a special kind of written language, with particular grammatical rules, vocabularies and linguistic conventions. Rather than being printed and read by people, code *runs*, it is a *self-executable* language. It can make decisions and can make or effect material change – causing a switch to close, a valve to open or operating a servomotor. Importantly in reaching these decisions, it can evaluate available data and take an action automatically without human oversight. In a sense then it can be argued that code exhibits emergent properties with aspects of complex behaviour beyond conventional electro-mechanical devices. As Thrift and French (2002: 310) note software exists "somewhere between the artificial and a new kind of natural, the dead and a new kind of living"; it often has a "presence as 'local intelligence'". Again thinking about the mobile phone in your pocket as a dense package of complex code that works as a 'local intelligence' it has a sense of a life of its own, it is doing things for you and for itself without being instructed to do (polling the telecommunications network, checking signal strength, monitoring battery life, storage, waiting for messages, updating, self-configuring, and so on).

Main Entry: code
Pronunciation: 'kOd
Function: noun
Etymology: Middle English, from Middle French, from Latin caudex, codex trunk of a tree, document formed originally from wooden tablets
Date: 14th century
1: a systematic statement of a body of law; especially : one given statutory force
2: a system of principles or rules <moral code>
3
a: a system of signals or symbols for communication
b: a system of symbols (as letters or numbers) used to represent assigned and often secret meanings
4: genetic code, Date: 1961, the biochemical basis of heredity consisting of codons in DNA and RNA that determine the specific amino acid sequence in proteins and appear to be uniform for all known forms of life
5: a set of instructions for a computer

- code·less /-l&s/ adjective
Function: verb
Inflected Form(s): cod·ed; cod·ing,
Date: 1815,
transitive senses: to put in or into the form or symbols of a code
intransitive senses: to specify the genetic code <a gene that codes for a protein>
- cod·able /'kO-d&-b&l/ adjective,
- cod·er noun

(From Merriam-Webster Online Dictionary, 2003, <www.merriam-webster.com>.)

The reason why software matters is its ability to do work in the world. This is most apparent from the ways that code can distribute, generate, monitor and control data exchanges across a range of media (reading and processing data to and from memory and discs, transmitting over cables and wirelessly through the air). Data is the lifeblood of code and seems to accrete even without human effort, programs generate temporary files, systems log events, messages are received and stored, application updates are downloaded and media is streamed from different sources.

Theorising code

In the last decade or so the field of Software Studies has emerged at the intersections of digital art, media theory, hacker intelligentsia and social science scholarship. It seeks to create an expanded understanding of software that extends significantly beyond the technical. The focus here is on developing cultural and theoretical critiques of how the world itself is captured within code in terms of algorithmic potential and formal data descriptions. A leading theorist in the field, Lev Manovich (2008: 6) states: "I think that Software Studies has to investigate both the role of software in forming contemporary culture, and cultural, social, and economic forces that are shaping development of software itself." As such it calls for transdisciplinary research methods and Matthew Fuller (2008: 2) has argued that it "proposes that software can be seen as an object of study and an area of practice for kinds of thinking and areas of work that have not historically 'owned' software, or indeed often had much of use to say about it." There is much, more I believe, that needs to be said by social scientists and urban researchers who have traditionally not had much to say about the spatiality of software beyond more generalised critiques of computerisation and the inequalities in IT provision and access to the internet.

A number of geographers have recently advanced a range of conceptual ideas and practical strategies to begin to understand how the diversity of software's agency contributes to the production of city space. This research is founded, in part, on a key 2002 paper by Nigel Thrift and Shaun French that set out the nature of software as a having the capacity to *automatically* produce space and thus have "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." They began to document the extent to which Euro-American societies are "…interwoven with computer software" (2002: 309). Working from this premise, Rob Kitchin and myself sought to unpack how software can automatically beckon space into being in our 2005 paper. We did this by firstly defining the range of forms of software into a four-level hierarchy: (i) individual coded objects, which can be linked to form (ii) coded infrastructures, that are monitored by and also transport (iii) coded processes. Coded objects, infrastructures and processes are in turn, combined together to form larger (iv) coded assemblages. This hierarchy enables the software, through its varying degrees of technicity (power and productive capacity for work) to transduce space, that is it brings new spatial formations into existence to solve a problem or perform a task. We elaborate on the nature of these spatial transductions through the enrolment of software, arguing there are three distinct levels of transductions: (i) code/space, (ii) coded space, (iii) background coded space. In the first level of transduction the technicity of software is so significant that the space brought into being *depends* on the operation of the code. There is a dyadic relationship between the space and the code – hence the co-joint nature of our term 'code/space' – and if the software fails to operate then the space is not produced. In the second level of 'coded space' the transduction is mediated by code but the relation is not dyadic so if the software were to fail to operate for whatever reason the space would still be produced as intended to solve a problem or perform a task. However, the nature of the spatial transduction without software is potentially a less efficient solution to the problem or a more costly way to perform a task (e.g., failure of computer system forces workers to use a 'manual' backup procedure that is much more labour intensive). The lowest level is a transduction-in-waiting so to speak, what we term 'background coded space', is when code exists and has the potential to mediate a solution if activated. Much of ordinary living in Western cities occurs in 'background coded space' where people are surrounded by coded objects, coded infrastructures and coded processes that can be called upon in myriad of ways to solve a problem or perform a task.

The social implications of the spatial work software has been theorised by Stephen Graham (2005), who extended the well known notion of 'social sorting' from surveillance research to argue that code is increasingly capable of automatically sorting spaces and attendant individual access and social practices. Graham developed the concept of 'software sorting' which is useful as it focuses attention on the "central role of computerised code in shaping the social and geographical politics of inequality in advanced societies" (2005: 562). The algorithmic ranking, classification and decision-making of software is increasingly being applied to drive evermore more profitable consumption and also in the name of urban securitization code is " …now being widely applied in efforts to try to separate privileged and marginalized groups and places" (Graham 2005: 562). It is politically significant that the nature of software sorting be investigated and potentially challenged by critical urban scholars and activists because they represent a new form of automated discrimination that is largely "… invisible from the point of the users, [where] prioritizations are often not evident either to the favoured groups or places or to the marginalized ones" (Graham 2005: 566).

In addition to potentially discriminatory sorting of social spaces, code has also been theorised in terms of the ways it can simulate future spaces and thereby regulate how they come into being by what has been termed 'anticipatory governance'. For example, in Peter Adey's work on mobilities and surveillance practices, he shows how the orderliness of flows through spaces often depend on how "software simulations make the future present and actionable-upon by alerting the users to future possibilities" (Budd and Adey 2009: 25). Again these algorithmic processes operate, often, without scrutiny or questioning of the basis of their underlying model of reality with its artificial parameters and missing variables, and the unacknowledged implications when the 'what-if' scenarios generated are applied in contingent live situations; as simulation models "move into the public domain their inherent uncertainties and qualifications may be forgotten and the public seduced into accepting their 'crystal ball' like assumptions" (Budd and Adey 2009: 8). The scope for code in simulation models to work in an anticipatory fashion, particularly in the domain of surveillance and governmentality, has social implications. The predictions of the future, created algorithmically and automatically by code, do work in the world to *prevent* that future scenario from coming into being. Such pre-emptive mechanisms have much appeal in the risk-conscious and real-time world of global mobility, but they clearly raise serious issues of ethics and power. In space-times where anticipatory governance using software simulations is active, how can people be sure of the social equity in the design of the code that affects, very materially, their life chances?

Exemplifying code

"The modern city exists as a haze of software instructions. Nearly every urban practice is becoming mediated by code."
(Amin and Thrift 2002: 125)

Code in the city can be studied empirically in a number of distinct domains. An obvious place to start, building in part on the work of Adey and Graham, is to focus on processes of securitisation and networked surveillance in which software is enrolled to extend coverage and try to automate analysis and enactment of governance. Code/spaces have become essential to the ongoing consumption practices in the cities, for example most retail services and logistical supply chains depend on coded networks, databases and automated software systems to ensure their smooth operation. Code has also permeated many home as it becomes crucial to solving many domestic tasks (cf. Dodge and Kitchin 2009). Here I want to focus attention on movement through cities and particularly on road systems and car driving.

This entails thinking about the spaces of circulation and also the places of being stationary, the varying speeds of vehicles, mechanical failures and illegal behaviours, issues of congestion and the adaptability of infrastructures over different time periods (hours, day, weeks, years). Such complex and dynamic issues are at the centre of the planning, management and living in contemporary cities. Investigation of these themes around movement and driving also provide linkages to novel work across the social sciences under the rubric of the 'mobilities' paradigm that focuses on the "analysis of different forms of travel, transport and communications with the multiple ways in which economic and social life is performed and organized through time and across various spaces." (Urry 2007: 6).

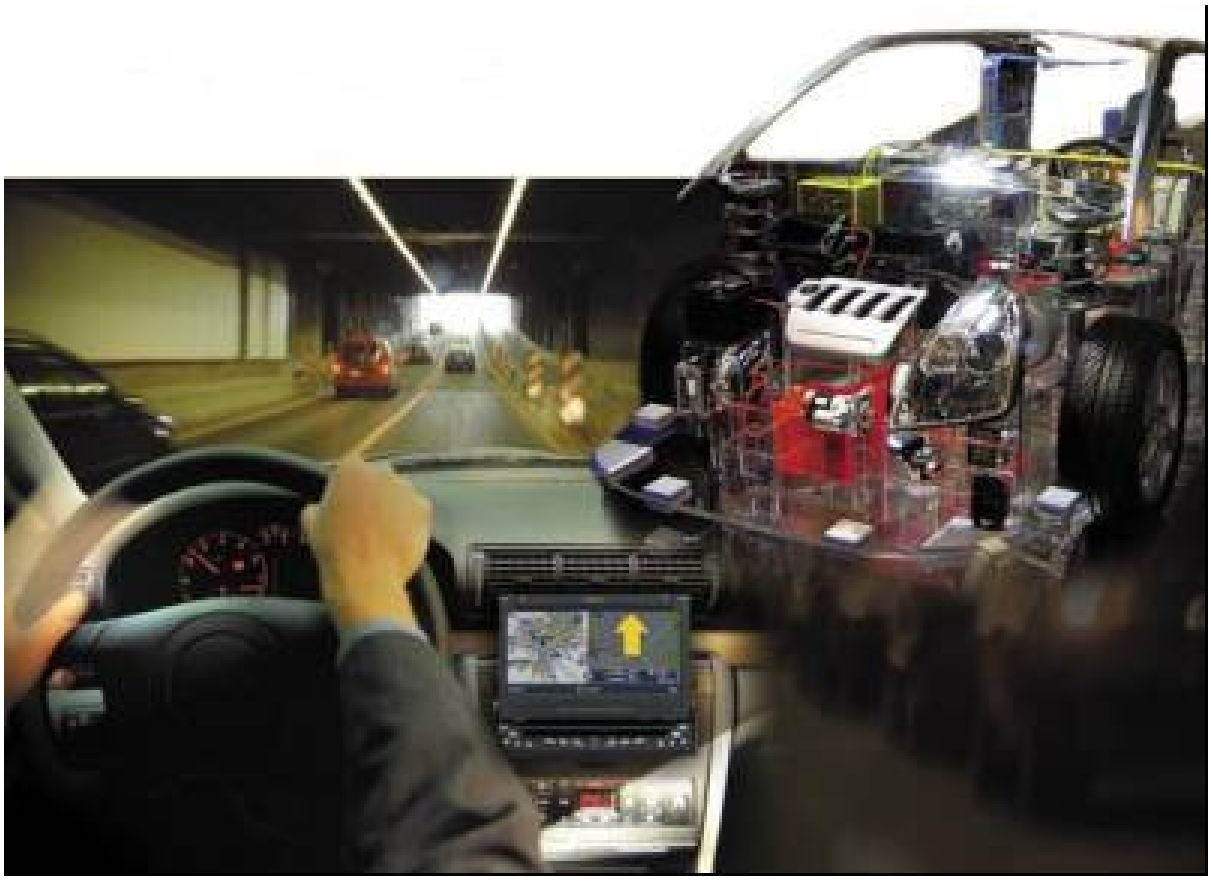
A defining characteristic of the modern city is the dominating presence of the automobile, with the daily practices of many people revolving around the extended and individualised mobility they afford and myriad urban spaces are configured to fit their bulky physical form and speed of movement.

Here I want to consider the degree to which these driving spaces are now being transduced by the technicity of software into coded space or code/spaces.

The Car and Code

To begin at the immediate scale of the car a good case can be made that they have been thoroughly recast in the last decade as code/space through externalised software processes that represent them and the internal embedding of code to augment the electro-mechanical working of the vehicles themselves. All new vehicles are now conceived within software environments, their material forms being sculpted in 3D modelling applications and their engineering requirements and mechanical parameters being virtually tested and refined within CAD systems and other specialised software tools.

Figure 1. Cars are conceived, designed, manufactured and distributed in software.
(Source: eSafety project flyer, 2005:2)



Manufacturing cars is sophisticated, exacting and highly automated, taking place in streamlined plants with computerized and robotic assemblage lines. These run as lean production systems in which thousands of individual components are drawn in from globalised supply chains enabled by networked information systems and electronic data exchanges. Many cars are built on demand to meet specific customer orders which are held in software databases that parallel the material vehicle. When the finished cars are leased or sold all the owner and vehicle details are held in a range of databases and customer relations management systems, including those for registered keeper, taxation status, road worthiness testing and mandatory insurance. Vehicles in legal and legitimate ownership are permanently tracked by a virtual data trail maintained in large, anonymous databases.

Automobiles are products of code/space but increasingly software is becoming bound into the very materiality of the vehicles themselves. Consequently, when they are driven they operate as mobile code/spaces. As Thrift (2004: 50) notes, "[a]lmost every element of the modern automobile is either shadowed by software or software has become …. the pivotal component." The calculative power of code tunes mechanical performance, augments conventional electrical systems, adds new functionality and, most significantly, it supplements the cognitive abilities of human drivers.

In many respects the outward appearance of cars has changed little in the last decade, but in terms of how they operate contemporary cars are really a collection of computers on wheels. A host of electronic control units (ECUs) monitor, mediate and modulate all manner of mechanical aspects of the vehicle. This is particularly so in luxury models where the addition of evermore sophisticated software systems is seen as a key element of product innovation and differentiation from competitors. However, even in mass-market, basic model vehicles code is being enrolled routinely as a core component. It can be argued, therefore, that cars on city roads represent one of the densest concentrations of digital computation and embedded software that most people encounter in the course of their everyday activities.

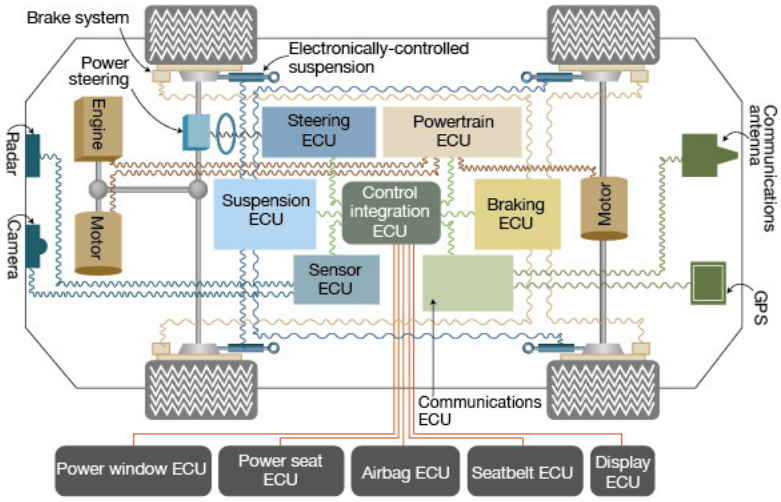


Figure 2. Schematic of typical vehicle management systems: 'computers on wheels'. (Source: Kariatsumari K, (2005) "Packing more electronics into cars", Nikkei Electronics Asia, September, page 30.)

Elements of software in the car are at least partially apparent from the now common presence of digital displays on the dashboard. Another indicator is the changed operation of some controls, such as the switch from mechanical locks to radio based keyless entry and remote locking systems that require authentication from an embedded ID code. Yet much of the coding up of the driver is not visible from the superficial inspection of vehicle controls because most software-enabled systems that envelope and regulate their actions do so surreptitiously so as not to undermine the 'driver experience' and their belief that they are in control. However, there are a growing range of driver-assistance systems depending on software in cars aimed primarily at increasing the safety of the vehicle and occupants. (Other coded systems are focused on enhancing convenience and vehicular performance, e.g., automated engine optimisation for fuel economy, logging usage to aid servicing). The technicity of code works to transduce the car and thus the spatial

performance of driving in at least four domains:

(i) reduce the cognitive burden on drivers (e.g., turn-by-turn voice navigation instructions from satnav making navigation easier)

(ii) reduce the level of kinaesthetic and spatio-perceptive skills required (e.g., distance detection within parking aids)

(iii) reduce the physical strength/endurance needed to drive (e.g., active steering, active cruise control)

(iv) sense environmental conditions beyond normal human senses (e.g., black ice detector).

The major hazard for drivers and vehicle occupants is a crash. A raft of 'safety through software' systems are now available in cars or are under active development by the automobile manufacturers. These augment the well established physical and mechanical safety measures. They have been likened to an 'electronic crumple zone' (Economist 2008: no pagination) where "if a collision seems likely a warning is given. When the driver puts his [sic] foot on the brake pedal the system automatically applies the optimum pressure required to avoid hitting the car in front. If the driver fails to respond, the brakes come on automatically." Code is thus enrolled in a significant sense to reduce the risks of a crash event by aiding the driver's road awareness and potentially intervening before the driver reacts (e.g., active breaking). If a crash does occur other software systems can help mitigate the immediate effects and also automatically summon assistance to the correct location.

The underlying assumption is that the driver is most often the causal 'problem' in an accident event and, in some senses, need to be protected from themselves. The fallibility of human judgement and lack of attention can be compensated, to varying degrees, by the technicity of code. Sometimes it may be appropriate for software to actively overrule driver's intentions or to act without driver authorisation if the algorithm detects risk above threshold limits. This a radical change in the way a vehicle's controls work, with a shift away from direct physical connections

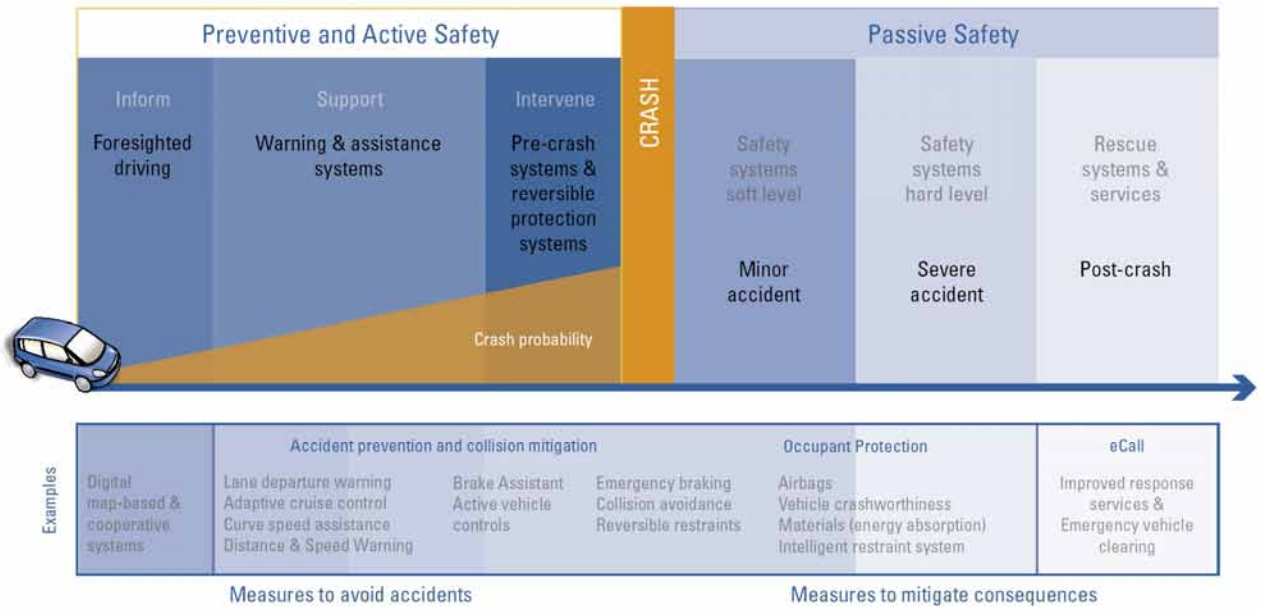


Figure 3. eSafety system and technologies concept diagram. (Source: eSafety project Fliyer, 2005:10).

between the driver's embodied actions and mechanical response to software-mediated 'drive-by-wire' operations. So, physical pressure on the break pedal does not engage the car's breaks but signals to the software in the breaking ECU. This code interprets your intentions algorithmically, and in combination with other sensed data about the car's speed and stability, it then decides how much mechanical breaking can be safely applied to the wheels. Important aspects of driving are superficially unchanged but actually now *depend* on correct operation of software the car has been transduced into a code/space.

Street as Software Systems

Figure 4. View of a control centre for a road network. The primary interfaces of monitoring and control software are evident on the numerous screens dominating the working space. (Source: Midland Expressway Ltd, <www.m6toll.co.uk>.)

Extensive urban road infrastructure of tarmac, conventional signs, and traffic controls in the form of pre-set traffic lights, fixed tolls and solid pollards, are rapidly being complemented with 'smart media' - digital, networked infrastructures controlled by software - that aim to more effectively monitor and regulate the street system in real-time. Examples of code enrolled in active traffic management include the automatic altering of traffic light sequences and the updating of road speed signs, automatic logging of vehicular congestion and variable toll charges, and networked speed, red light and bus-lane cameras designed to discipline driver behaviour. These software-enabled technologies, when used in combination, aim to produce wide area of intelligent transport systems that make more efficient use of roads. As such, street systems are 'coded space' although in most cases the relationship between them is not one of dyadic dependency – if the software fails the roads still come into being as drivable road space but perhaps less efficiently or safely.

Traffic volumes are continuously monitored drawing upon a range of software controlled sensing infrastructures distributed at strategic points across the city (induction loops, infrared cameras at traffic lights, networked video camera's that can count passing vehicles from their number plates). The data feeds into urban traffic control centres that simulate the traffic levels in the near future within software models and can adapt control systems, such as timings on traffic light phases to try to minimise congestion and smooth flow to provide more consistent journey times for majority of drivers.

The telematic monitoring of individual vehicles, enabled by software, opens up the possibility to identify and continuously track the movement of all cars across the city in real-time. This capability to transduce the streets into a continuous code/space will likely have significant implications for the costs of driving in terms of congestion charging, road pricing and variable insurance rates. Conventionally drivers pay a fixed annual tax and insurance premium for legal opportunity to use the road system as much or as little as they want. Code will enable much more flexible and dynamic pricing models, with more intensive or riskier drivers paying more. For example, a number of insurance companies are experimenting with onboard tracking devices that generate vehicle movement data for software to dynamically calculate insurance premiums that reflect driving patterns (kilometres driven, routes, time of journeys) behaviour and the different types of locations in which they park their vehicles.

Figure 5. Examples of coded infrastructures enrolled to monitor and control traffic flows in Los Angeles. Picture left: shows black swirls of induction loop under the road's surface that can detect stationary cars at the junction. Picture middle: a bland control box that adapts the phasing of sets of networked traffic lights according to predict flows. Picture right: a steerable video camera with ANPR. (Source: The Center for Landscape for Land Use Interpretation, <www.clui.org/clui_4_1/ondisplay/loop/exhibit/>.)

Forms of congestion charging that charges drivers for entering specific areas of a city or more continuous road pricing schemes that cover whole journeys are increasingly likely to be deployed by governments looking to raise revenue by what could be perceived as a progressive 'green tax'. London has led the way with its charging zone in the centre, in operation since 2003, that relies on automatic number plate recognition software to identify vehicles and verify if they have paid the required fee or not. Failure to pay is flagged in the database automatically and a fine is generated. In 2008 a more sophisticated congestion charging scheme was proposed in Greater Manchester with two control rings and variable pricing depending on the direction of journeys and time of day. Such a complex format of charging for tens of thousands of daily journeys is only feasible with wholesale automatic governance through code. The proposal in Manchester would have put up virtual barriers of code/space that would have transduced every vehicle passing through the sensors and although defeated in a popular referendum (in December 2008), it seems likely that an adapted scheme will return in the near future as the planners and politicians look for a technological fix to 'excessive' automobility in the city.

Some car rental companies are now using telematic tracking systems to monitor where rental drivers take the vehicle, with penalties imposed if the car is taken to somewhere outside of the rental contract (e.g., across a border or off-road). Other systems are sold as products to parents so as to monitor the location of 'at risk' teen drivers. For example, Omnitrack, designed as an anti-theft device, allows parents to track in real time where a child's car is and how fast they are travelling. electromechanical tachographs that regulate drivers hours.

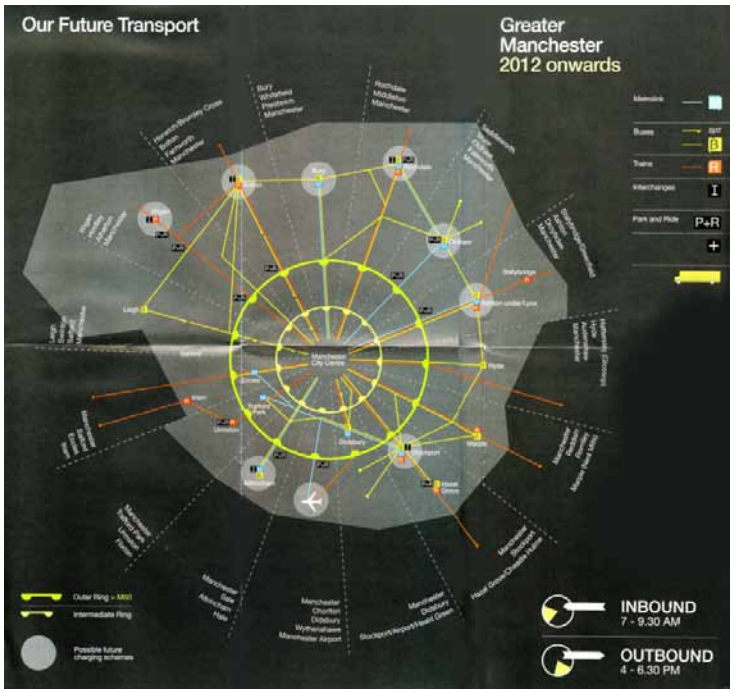


Figure 6. Diagrammatic view of the proposed congestion charging scheme for Greater Manchester. (Source: GM Future Transport leaflet, 2008.)

It can also be programmed so that the company will contact the parents if any set parameters (e.g., speed or distance) are exceeded. A range of distanciated driver management systems, similarly using GPS and telematic tracking, are also becoming more common across commercial vehicle fleets. These monitor the behaviour of drivers operating commercial delivery vehicles, taxis, buses, emergency vehicles, and so on, and supplement electromechanical tachographs that regulate drivers hours.

Implications of a city as code/space

The work of code in contemporary cities is significant and given current social, economic and technical trends, seems set to expand greatly in the coming decade. This will bring benefits, offering opportunities, facilities and urban services for many and also reduced costs for institutions and companies delivering them. However, it will also bring an expanded range and magnified degree of risks from greater techno-social complexity in managing city functions that become *dependent* on code. These code/spaces are a risk because they are imperfectly understood, the software is often poorly engineered, hard to diagnose when it misbehaves and difficult to fix when it fails. The ways that parts of city infrastructure and facilities fail will not only be through observable physical faults, there will be new vulnerabilities resulting from incorrect data or errors and unanticipated conflicts from new or updated software components. As code/ spaces become common place we will all have to get used to software errors in many more areas of daily life and learn to cope when the code crashes.

At the level of individual practice, the growth of software will have significant implications as it mediates and regulates more and more everyday activities, like driving. Software can be read critically as threatening to accepted notions of personal privacy, individual autonomy and social equity. For example, in the scenario of dynamically priced insurance rates automatically calculated by software on driving patterns and mandatory road pricing requiring real-time tracking of all journeys. In both cases, software enabled technologies seek to enforce differential access on the basis of certain criteria, usually authorised identity or ability/willingness to pay, and thus ensure that the road system is segmented; those who are entitled have access to the right parts of the system and those who do not are excluded. Of concern to some commentators is that financially based, software-driven 'social sorting', works to benefit affluent drivers while penalising the poor and those classified as higher risk, either by denying them access to a section of road or area, forcing them to take more expensive routes in terms of time and distance, or by having to pay higher premiums ('discrimination-by-postcode' where poorer areas tend to have high premiums due to higher crime rates). Such sorting thus works to further marginalise and exclude poorer sections of society from essential urban infrastructure. It is therefore essential that urbanists and social scientists should focus attention on describing *where* code is working in cities, account for *how* it works and offer explanations of *whom* it works for.

Figure 7. In regulating driving the ultimate form of governance is through physical barriers that can regulate access. Examples include car parking barriers and automated bollards that will lower to permit authorised vehicles to pass but rapidly raise to block everyone else. (Source: <www.youtube.com/watch?v=BwnfeDtnuds>.)

References

Amin A, Thrift N, (2002) *Cities: Reimagining the Urban* (Polity Press, Cambridge).

Budd L, Adey P, (2009) "The software-simulated airworld: anticipatory code and affective aero mobilities", *Environment and Planning A* 41(6) pp. 1366-85.

Economist (2008) "Stopping in a hurry", *The Economist* 11 December <www.economist.com/science/displaystory.cfm?story_id=12758720>.

Fishman C, (1996) "They write the right stuff", *FastCompany Magazine*, December, page 95 <www.fastcompany.com/online/06/writestuff.html>.

Fuller M, (2008) *Software Studies: A Lexicon* (MIT Press, Cambridge, MA).

Graham S D N, (2005) "Software-sorted geographies", *Progress in Human Geography* 29(5) pp. 562-80.

Kitchin R, Dodge M, (2010) *Code/Space: Software and Everyday Life* (MIT Press, Cambridge, MA).

Manovich L, (2008) *Software Takes Command,* 20 November version, <http://lab.softwarestudies.com/2008/11/softbook.html>.

Mirapaul M, (2003) "Deliberately distorting the digital mechanism", *The New York Times*, 21 April.

Thrift N, (2004) "Driving in the city", *Theory, Culture & Society* 21(4/5) pp. 41-59.

Thrift N, French S, (2002) "The automatic production of space", *Transactions of the Institute of British Geographers NS* 27 pp. 309-35.

Urry J, (2007) *Mobilities* (Polity Press, Cambridge).

Further reading

Papers available to download from <www.cybergeography.org/martin/>:

Dodge M, Kitchin R, (2009) "Software, objects, and home space", *Environment and Planning A* 41(6): pp. 1344-65.

Dodge M, Kitchin, R, (2007) "The automatic management of drivers and driving spaces", *Geoforum* 38(2): pp. 264-75.

Dodge M, Kitchin R, (2007) "'Outlines of a world coming in existence': Pervasive computing and the ethics of forgetting", *Environment and Planning B: Planning and Design* 34(3): pp. 431-45.

Dodge M, Kitchin R, (2005) "Codes of life: Identification codes and the machine-readable world", *Environment and Planning D: Society and Space* 23(6): pp. 851-81.

Dodge M, Kitchin R, (2005) "Code and the transduction of space", *Annals of the Association of American Geographers* 95(1): pp. 162-80.

Urbis Research Forum Review
Vol. 1, Issue 2
2009
ISSN: 2042-034X

Urbis
Cathedral Gardens
Manchester
M22 4FP