11 A Manifesto for Software Studies

The longer a system has been running, the greater the number of programmers who have worked on it, the less any one person understands it.
—Ellen Ullman

You think you know your computer, but really all you know is a surface on your screen.
—Annette Schindler

More work is needed on understanding computer code and our current tools and methodologies are limited in trying to unpack its production, meaning, circulation and reception.
—David M. Berry

In this book, we have offered an analysis of software and its role in the unfolding practices of everyday life. We have not sought to provide a theory as to how computing and computation should be developed; rather, our aim has been to detail a set of ideas for thinking about software and its relations—how it is constitutive of, situated in, and does work in, the world. Strangely, this type of analysis has only begun to occur in the last decade or so, with the nascent formation of software studies. It is true that there has been plethora of research and literature about software, and the formation of new disciplines and fields such as computer science, human–computer interaction, artificial intelligence, Internet studies, information science, and software systems engineering, which have mirrored the development and growth of software-enabled technologies, systems, and products. And yet, these works are strangely silent in many respects, tending to focus on the development of software from an engineering-centered perspective and the social, economic, political, and cultural impacts of software-enabled technologies, as opposed to the software that enables such technologies. Where scholarship has examined in some depth the relations between software and people, it has tended to concentrate on individual cognitive and ergonomic aspects, drawing in particular from psychology and health-related disciplines, along with media research focused on the conceptualization of software in terms of social

communication or economic transactions. Software studies differs from these other fields in that it focuses analysis explicitly on the conceptual nature, and productive capacity of software, and its work in the world, from a critical social scientific and cultural perspective. As Manovich (2004, 6) notes, "if we don't address software itself, we are in danger of always dealing with effects rather than causes, the output that appears on a computer screen rather than the programs and social cultures that produce this output."

By concentrating attention on the code itself, software studies seeks to create a theoretically and empirically rich understanding of software and its radically diverse constitution and growing contribution to social life. Rather than focus purely on the technical, it fuses the technical with the philosophical to raise questions about what software is, how it comes to be, its technicity, how it does work in the world, how the world does work on it, why it makes a difference to everyday life, the ethics of its work, and its supporting discourses. Software studies then tries to prise open the black boxes of algorithms, executable files, captabase structures, and information protocols to understand software as a new media that augments and automates society.

Code/Space and the literature we have drawn upon represent an initial foray into conceptualizing software as a vital source of social power. *Code/Space* further provides philosophical and analytical tools for making sense of code. Our aim has been to add to this growing body of knowledge in a general sense, considering in detail the nature of software, the relationship between code and objects, the effects of code on changing modes of governance, and how code can engender novel creative practices. We are also interested in exploring how code offers new possibilities to empower people. In so doing, we have argued for an ontogenetic conceptualization of software that works to destabilize the notion that code has a stable ontology, instead conceiving of code as contributing to unfolding practices; its work always in the process of becoming. Such a move recognizes that software is both a diversely created product and a key producer of social relations, with the relationship between software and society understood as a hybrid assemblage that is contingent, relational, productive, and made in the moment. In particular, we have argued for the creation of a set of knowledge that takes space, as well as time, seriously. Software is thoroughly spatial in its production and work, transducing complex spatialities.

Given the nascent state of software studies, there is much more work to be done to extend, deepen, and refine the analytical tools available for researchers and to explore and examine the various facets of software and everyday life. In the rest of this chapter, we set out a brief manifesto for the kinds of work we believe necessary in the coming years. This manifesto is inevitably preliminary and partial, framed by our own knowledge and ideological vision of the world. It is therefore also an invitation to others to reenvision and extend.

How Code Emerges

It seems to us that software is largely understood from an engineering and organizational point of view. Social analyses tend to focus on the consequences of computerization, rather than how software emerges and does work in the world. And yet, software is not simply a technical device. Software is both a product and a process. As we detailed in chapter 2, software is the product of a sociotechnical assemblage of knowledge, governmentality, practices, subjectivities, materialities, and the marketplace, embedded within contextual, ideological, and philosophical frames, and it does diverse and contingent work in the world.

In designing and writing software, developers make, on the one hand, critical, ontological decisions about what to capture, categorize, and represent in the world. On the other hand, developers make epistemological decisions about the relations between capta and how they should be processed to beckon into being actions in the world. Developers often unconsciously place a particular philosophical frame on the world that renders it amenable to the work of code and algorithms, thus realizing a specific system of thought to address a particular relational problem. As a consequence, the consistent and automated generation of precise forms of capta has become a tremendously important business because it provides the raw material that code works upon, often transducing it into information. But what does it mean for society when the capta and apparatus that underpins critical decision making is based on a constricted set of criteria designed to satisfy the limitations of algorithmic processing? What are the implications of reducing the world to a small ontological subset and a sequence of algorithms? Does the sensibility of a relatively small cadre of programmers become the overriding blueprint for future everyday social relations? Will defaults in code become the defaults for living? And what are the consequences of algorithmic processes that are so complex and opaque that even their designers are unsure as to how an outcome was arrived at (which seems to be the case for some pattern recognition software used in passenger profiling)? Indeed, unpredictable outcomes and unforeseen circumstances are a major cause of software glitches and system failures.

While we and others have made a start in exploring the nature of software, there is much research and appraisal to be done in terms of working through how to conceptualize code and its work. It seems to us that there is a critical need for more detailed ethnographic studies of how developers produce code, and the life of software projects, in order that we can build a better understanding of the ways in which software is diversely created and unfolds within contextual frames. At present, there are only a handful of in-depth studies that examine the complexities of how code is produced as a collaborative manufacture between programmers with diverse subjectivities, abilities, and worldviews, and corporations and institutions with particular

philosophies, ambitions, and resources, and shaped by the market and the vagaries of investment finance. There is not enough research on how code is scripted through collective cycles of drafting, editing, compiling, and testing, or how it is produced in relation to a specification that is constantly being tweaked by programmers, shaped by outside audiences, and consumed by the public. There is also a need to develop a subarea of software studies—algorithm studies—that carefully unpicks the ways in which algorithms are products of knowledge about the world and how they produce knowledge that then is applied, altering the world in a recursive fashion.

An example of this type of research is Mackenzie's (2009) analysis of an esoteric but essential algorithm known as Viterbi that provides the guts of digital signal processing essential to the operation of cell phones, along with a growing array of other wireless devices that are at the vanguard of pervasive computing. He applies the Deleuzian notion of "envelopment" to conceptualize how this algorithm creates calculative spaces characterized by change that is always changing, what he calls "intensive movement." In other words, these are types of spaces in flux that cannot be mapped in certain terms, but can only be guessed at in probabilistic ways; they are spaces existing more in an unseen quantum universe than in the experienced fixity of Newtonian space. The enveloping radio spaces swirling around, between, and through the spaces populated by people's cell phones exhibit this intensive movement with ever changing patterns of congestion and contestation between signals that inevitably overlap, disrupt, and inhibit each other. Yet the Viterbi algorithm is able to make sense of the intensive movements in real time because it "assume[s] that we can only hope to determine the most probable series of sent signals" (Mackenzie 2009, 1303) which is at odds with "the images of strict determinism sometimes associated with digital technologies" (p. 1304). The work of this algorithm, which is now largely taken for granted, means that the phone in one's pocket receives only one clear connection, despite being in the midst of a cacophonous tumult of competing, continuously changing signals. In short, the Viterbi algorithm is creating the fundamental conditions in which communicative practices (a phone call or text) takes place.

How Code Performs

Code does work in the world. It has has technicity; it divulges and affords new kinds of automated agency, opening up new possibilities and determinations. Software is an increasingly capable actant performing in the world; it makes a difference to how people solve problems in a variety of intended and unintended ways, unfolding contingently and relationally with respect to prevalent conditions and contexts. While there has been voluminous research examining how computer technologies and digital infrastructures are reshaping everyday life, there has been relatively little focus

on the role of software in such reconfigurations. That is not to say that what has been done is not useful and valuable. We simply find it surprising that there has not been much more research specifically focused on the messy and detailed enrollments and effects of code itself given the extent to which coded objects, infrastructures, and processes are ever more folded into everyday life, creating new coded assemblages. As a result, we believe that a core agenda for software studies is to produce detailed case studies of how software does work in the world, and to develop theoretical tools for describing how and explaining why, and the effects, of that work.

In this book, we have undertaken some broad brushstroke analysis with respect to how software is reshaping travel, home life, and consumption, but the kinds of study we are envisioning would be significantly more comprehensive and systematic pieces of work based on extensive fieldwork and empirical analysis that would tease out in detail the contextual ways in which code reshapes practices with respect to industry, transportation, consumption, governance, education, entertainment, and health. It would also need to be sensitive to place and scale, cultural histories, and modes of activity (for example, a study comparing the effects of code in rural Ireland and urban Manchester). Such in-depth code studies will provide a richness of observation and insight for scholars to better understand and explain how code makes a difference in those contexts. Again, we have sought to provide some initial theoretic insights and interpretations. We have argued that code makes a difference to the nature of objects because it imbues them with the capacity to do additional and new types of work. We have noted that code alters everyday spatiality because it has technicity to alternatively modulate how space unfolds. In addition, we have explored how code reconfigures governance because it enables new modes of regulation and automated management. We have further highlighted how code creates new forms of knowledge production, creative practice, and processes of innovation. We would consider these to be preliminary readings that require further refinement; initial forays to make sense of code's work. There is clearly much more to be done to sharpen and extend such interpretations, or to replace them with more nuanced and sophisticated insights.

How Code Seduces and Disciplines

We have made the case that code is rapidly moving to a state of pervasiveness in some aspects of daily life because it both seduces and disciplines people. Code interpellates people to its logic, wherein they voluntarily and willing submit to the agency of software. In fact, they often desire and embrace the framework of software, because it offers them real benefits with respect to convenience, efficiency, productivity, flexibility, and creativity. In short, software has the capacity to make society safer, healthier, richer, and more enjoyable. At the same time, code disciplines people by making them

enact certain grammars of action and enforcing more pervasive modes of surveillance, automated management, and self-disciplining. People thus offset new forms of regulation against the benefits gained. As such, many consumers willingly trade aspects of their privacy for a service in the name of their own empowerment (Andrejevic 2007). Of course, such a trade works so long as consumers are aware that a trade is occurring and do feel empowered, the terms of service remain favorable, there is choice that enables them to transfer to another service, or people are content to live with the consequences of heightened forms of regulation. In some arenas, such as air travel and identity cards, this balance is constantly being renegotiated as the regulators push for ever greater compulsion to generate ever more capta.

One of the key factors in mediating this balance, and how people view and understand code, are the discursive regimes that surround it, and the work that it performs. There is a powerful and consistent set of discourses that promote and support the deployment of software across a whole series of domains. These discourses include safety, security, efficiency, anti-fraud, empowerment, productivity, reliability, flexibility, economic rationality, and competitive advantage. Some of these discourses gained significant robustness with increased securitization in the war on terror and the drive to stimulate economic growth through the creation of a so-called knowledge economy founded on software-enabled technologies. These ideas are challenging to counter because it seems counterintuitive to articulate a position that seems to be premised on being less safe, less secure, less productive, provide less choice, and more inconvenience. Such positions seem illogical and lacking common sense. That is not to say, however, that such discourses are not actively questioned and resisted by individuals, critical scholars, activist groups, and organizations, or that all parties promulgating such lines necessarily agree wholly on their aims or how certain goals are to be achieved. Indeed, it is often the case that there are significant internal differences between supposed allies, particularly where a software project cuts across competing agendas (for example, state security versus business flexibility).

To date, there has been relatively little research that has systematically examined how software seduces (or fails to seduce) people while simultaneously disciplining them. Further, there has not been enough detailed comparative exploration of the discursive regime of different software products and system configurations in separate locations (for example, Heathrow versus Dublin airport). Such research is necessary, we believe, if we are to more fully understand the role and effect of software in society. What is desirable are a number of in-depth case studies that examine how and why people adopt and submit to certain software products. This research should fully plot the complex and contingent ways that people understand and react to the discursive and affective fields surrounding software-enabled technologies. Further, this research should explore how people subtly balance the benefits of use against the negatives (the automated "gifts" associated with loyalty cards against profiling and targeted

marketing); and how they both use and resist software-enabled technologies (submit to airport security but simultaneously question the rules, and perhaps transgress procedures in some way). Of course, this social scenario is complicated because much of the work of code remains hidden from view and people are subject to its processing without fully knowing what is being performed. Even when people are knowingly subjected to algorithmic profiling (for example, at an airport when an immigration officer swipes a machine-readable passport) they are in a subordinate position and lacking details on how to challenge the decision.

In addition, there is a need to examine the ways in which discursive regimes are assembled over time by a variety of vested interests—government, corporations, and civil society—through different forums such as advertising, media coverage, online media, letters, statements, signs, and staff training. It is also important to piece together how the dynamic of different voices internal to the regime unfolds, and how the discursive regime is supported by legislation and quasi-legal conventions that legitimate the governmentality that code enacts. Other avenues to explore include how the discourses promoted are countered by those that question their logic and social implications, and how the interchange between these sides unfolds to shape how software is developed, deployed, and received. As illustrated by the debates over issues such as identity cards and a national identity database (in the UK), and electronic voting machines (in Ireland), the discursive landscape around software can change markedly over time, and just because the technology is available, does not mean that it will necessarily be employed.

How to Code Ethically

Code and the prospect of an era of everyware raises manifold social, political, and ethical questions, not least with respect to the regulation of everyday life and the alteration of the conditions through which life unfolds. While code has the authority to empower individuals, at its logical extreme, individuals may come to live in an almost fully panopticon society, with internal sousveillance reflected into intensive endogenous surveillance.

Whether such a situation comes to pass, only time will tell; but it is important to think through the ethics of code: how should the world be captured in code to minimize negative impacts, and how might code do work in the world that is beneficial to the largest number of people? What capta can be generated with respect to individuals? How should such capta be stored and processed? How should such capta be employed? Who should have the power to use such capta? What would it be like to live in a world without anonymity or privacy? What would it mean to live in a society where there is a permanent and lifelong record of all manner of actions, deeds, and misdemeanors? To date, these kinds of questions have largely been avoided by many

technologists because they are complex and difficult to think through and answer, but nonetheless these and related issues need to be examined in detail in order to consider how society and its spatialities should be managed and regulated. Elsewhere (Dodge and Kitchin 2007b) we have suggested that one path toward such an exploration is to construct an ethics of forgetting in relation to pervasive computing.

In such an ethics, we contend, the trend toward technologies that "store and manage a lifetime's worth of everything" should always be complemented by forgetting. Schacter (2001) details six forms of human forgetting, three concerned with loss, and three with error. Loss-based forgetting consists of transience (the loss of memory over time), absent-mindedness (the loss of memory due to distractedness at the time the memory relates to); and blocking (the temporary inability to remember—"it's on the tip of my tongue"). Error-based forgetting consists of misattribution (assigning a memory to the wrong source), suggestibility (memories that are implanted either by accident or surreptitiously), and bias (the unknowing or unconscious editing or rewriting of experiences). Schacter (2001) notes one other problem with memory—persistence, the recalling of events that would rather be forgotten.

Following the discussion in chapter 10, one can consider such forgetting in relation to ethical dilemmas posed by life-logging software whose developers seem to seek ubiquitous and "merciless memory" (Galloway 2003), and which seek to overcome both problems of loss and error in human memory. Life-log tools and software aim to produce a perfect digital record of events and activities that would not decay or fade; distractedness would be minimized through automated cross-referencing of life-log sources; and blocking would be minimized by intuitive retrieval and visualization capabilities. A life-log would minimize errors because the technology would not be open to misattribution, suggestibility, or bias—it would be an exact capta record of what the sensor witnessed and would not be open to reinterpretation and reworking. Moreover, the life-log would be augmented through the recording of detail beyond what an individual notices or knows. For example, each memory would be augmented by exact time-space coordinates, and possibly other environmental variables such as temperature, humidity, and physiological aspects such as heart rate. Moreover, it would add order, precision, completeness, multiple angles (taken from different sensor technologies to provide a multimedia memory), instantaneous recall of the whole archive, searchability, and filtering, and allow analysis (such as cross-referencing, charting of temporal development, producing value-added multimedia recollections, and plotting space-time patterns of activities) to what human memory or existing memory technologies (such as a photo album) can achieve. In other words, the life-log would not forget and would also augment through added detail.

Despite these qualities, which at first might appear to be significant benefits, we feel that the panoptic memory of such life-logs are highly problematic because they record without discretion. We see forgetting, therefore, not as a weakness or a fallibil-

ity, but as an emancipatory process that can free people from being swamped by excessive capta gathering and pernicious disciplinary effects. As Nietzsche suggests, forgetting will save humans from history (Ramadanovic 2001), because "forgetting turns out to be more benefit than bereavement, a mercy rather than malady . . . for no individual or collectivity can afford to remember everything" (Lowenthal 1999, xi). Forgetting allows people to be fallible, to evolve their social identities, to live with their conscience, to deal with "their demons," to move on from their past and build new lives, to reconcile their own paradoxes and contradictions, and to be part of society. This is why we suggest that a society with software systems that never forget and forgive has the potential to become a totalitarian regime.

Perhaps, in the process of designing and implementing more ethically orientated life-logging software, aspects of forgetting should be an integral part of any system. For us, this should happen from the bottom up and be a core feature of the life-log algorithms, rather than from the top down, wherein legislation or organizational policy is used to regulate "perfect" life-logs. So, rather than focus on the prescriptive needs for privacy protections to try to make everyware more ethically acceptable, we envision necessary processes of forgetting, following Schacter's (2001) six forms, that should be in-built into the code, ensuring a sufficient degree of imperfection, loss, and error. For us, this strategy of coding with ethics in mind would make the system humane and yet still useful.

Let us consider the case of a life-log of a journey across a city. Transience could be achieved by ensuring the fading or loss of details over time proportional to the length of time lapsed between capta generation and the present. Just as a person would simply start to forget parts of the journey, so the life-log captabase would gradually and subtly degrade the precision of the record with time.

Absent-mindedness could be ensured by having distractedness built into the sensing firmware of capta generation. The log would record the whole journey, but miss out certain pieces of capta because a recording media was switched off or was directed at something else.

Blocking could be incorporated at the time the life-log was being queried. At other times, the query could be answered with no problems.

Misattribution could be achieved algorithmically by the specific misrecording of part of an event, but not the whole event. For example, part of a journey would be randomly misattributed (having a coffee in Starbucks rather than Caffe Nero), but the overall journey in terms of traveling from A to B is correct. In other words, misattribution is meaningful in relation of time, space, and context. It is not the adding of false memories, but rather the tweaking of a past event.

Suggestibility could consist of the plausible rescripting of certain events after a particular time by software. Here, part of the journey would take a subtly different, but believable, route (taking street A rather than street B).

Lastly, bias could consist of rewriting all events based on pattern recognition; it could rescript the capta in line with past behavior, decisions, and preferences to create a record that would be consistent and plausible but subtly different. The journey could be an impression of the route rather than a perfect recording, highlighting the things seemingly more important; it would then become a memory and not a recording. Over time, the extent of suggestibility or bias could increase, adding a degree of uncertainty into the capta. Overall, then, a range of algorithmic strategies could be envisioned such as erasing, blurring, aggregating, injecting noise, perturbing data, and masking, that could be used to upset the life-log records.

While building fallibility into the system seemingly undermines life-logging, it seems to us that a fallible life-log, underpinned by an ethics of forgetting—an ethics that works at both the micro level (the individual level, being able to live with yourself) and the macro level (the collective level, being able to live in a society)—is the only way to ensure that people can forget, can rework their past, can achieve political growth and change based upon debate and negotiation, and can ensure that disciplining does not occur. Clearly, there is much scope for further normative thinking in terms of the fundamental design philosophies for everyware systems.

Full consideration of the ethics of code will also require other analyses that focus on the oversight and accountability of those employing code to enact forms of managerialism and regulation. To date, companies in the West such as cell phone operators and Internet service providers and content providers have mainly sought to use strategies of self-regulation to avoid overly punitive regulation by legislators. These are often accompanied by legal safeguards that determine what capta can be generated, stored, manipulated, used, and sold. As a result, data protection acts are already commonplace, as are legal standards regarding capta and its generation. However, given the range of new forms of capta such as life-logs, the potential misuses of such capta need to be charted, and the reach and effectiveness of such legislative control needs to be constantly monitored and amended accordingly. Indeed, the effectiveness of data projection laws has been limited for various reasons. These laws tend to be framed in ways inherently supportive of an institution's rights to collect data; compliance to regulation is often poorly policed; and enforcement for failures to comply typically do not relate to the individual damage caused. It is important that scholars undertake other such studies in normative ethics to fully consider how people want code as part of their lives. The alternative is, as Lessig (1999) notes, that code becomes law.

A Selection of Methods for Undertaking Software Studies

It would be unhelpful to be too prescriptive with regard to the detailed methodologies used to uncover the ways in which software emerges or how code works in the world.

Rather, we feel that a wide range of social science methodologies could be profitably brought to bear on the issue as appropriate to the philosophical lens used to understand the world. For us, that means using techniques that focus on the processes and practices of code, and which reveal its discursive and material constitution and effects. It also requires sensitivity to the issues of scale of observation necessary to capture the small moments of software transduction, and the hard-to-discern citational patterns of code. We are aware that in this book, we have mainly worked at the macro scale, mapping out at a relatively coarse level how software makes a difference to everyday life. This has been borne of necessity in order to provide a broad, accessible, and synoptic overview. There is, however, a pressing need for high quality analysis undertaken at the micro scale, mapping out the mundane and everyday practices of coding, and the embedding of software into everyday places. Analysis also needs to be undertaken into the ways that code mediates formerly analog activities including people's routine interactions, and how software vendors, the government, and other interest groups seek to promote and legitimate a coded approach to a diverse range of issues.

For us, the kinds of methods we would like to see applied include genealogies, ethnographies, observant participation, and envisioning using mapping and spatialization techniques. With respect to the first suggestion, we are interested in the creative application of a genealogical method from a Foucaultian perspective that seeks to trace out the contingent unfolding of a system of thought or set of actions rather than produce a rational, teleological historiography. In other words, we think there would be much merit in constructing as full as possible genealogies of the multiple, complex, and sometimes contradictory or paradoxical iterations of software projects—the evolution and contextual and contingent unfolding of ideas, decisions, constraints, actions, and actors that shaped their development. Through such an analysis, we can start to trace out over time the ways in which ontological frames emerge and become codified in algorithms, code has been produced, and software has seeped into everyday use.

Let us focus on the genealogy of algorithms. We believe it would be instructive to conduct a detailed archeology of how algorithms come to be constructed—to excavate the social lives of ideas into code—and how an algorithm then translates and mutates across projects to be reemployed in diverse ways. Such a genealogy would generate valuable insight into the coded production of knowledge; how concepts are molded into a capta ontology and translated from one medium to another; and how specific decisions at the point of programming influence the work the algorithm does.

The necessity of decoding the workings of obtuse algorithms at the heart of software systems, like cell phones, is difficult to achieve in ways that produce meaningful knowledge in a social science sense (Graham 2005, Mackenzie 2009). This is partly a problem of "black-boxing"—what Mackenzie (2009, 1299) describes as the

"submersion of algorithms into commodity hardware." The obscurity of the operational logics of algorithms is apparent in many offline and online settings (Zook and Graham 2007 presents a critique of search engine ranking algorithms in this regard). Yet this is not the only issue, because as Mackenzie (2009, 1295) notes: "the algorithmic processes . . . offer a strong challenge for research. . . . In their somewhat stunning complexity, they seem to bear only a tangential relation to the powerful dynamics of belonging, participation, separation and exclusion typical of contemporary network cultures."

Ethnographic studies provide an in-depth, holistic analyses of social phenomena describing the many relations between actors and the material world they occupy. Empirical material is usually generated by participant observation undertaken over an extended period of time (several months or more) and in-depth interviews with a wide range of stakeholders. These interviews are complemented by other techniques such as a hermeneutic reading of related documents and artifacts (such as policy reports, manuals, e-mail exchanges, visual materials, and work spaces) and time diaries. In essence, an ethnography seeks an immersive understanding of the lifeworld of a community—its social relations, its rhythms, its cultural meanings, its patterns of power and decision-making—in order to comprehend how it is constituted and how it continuously unfolds. An example relating to software development is that conducted by Rosenberg (2007). For a period of two years, he followed a start-up company as they sought to bring a software product to market, sitting in staff meetings, making lengthy observations of the workers interacting with each other professionally and personally, interviewing key actors, and just generally "hanging round the place" as the team plotted, argued, coded, reprioritized, downsized, expanded within an ever shifting context of new ideas, investor confidence, personality clashes, and staff turnover. In so doing, Rosenberg soaked up sufficiently rich details that he was able to provide a convincing holistic account about the nature of software development as a messy, emergent process (rather than the idealized sequence set out in software engineering textbooks). This kind of study needs to be extended to investigating the production and employment of software in different arenas and practices, such as home, leisure, work, travel, and consumption.

These ethnographies, which are necessarily small scale and tightly focused, would be well complemented by wider-ranging observant participation. This is a method that we have employed extensively, examining the ways in which we ourselves interface with coded objects, infrastructures, and processes as a means to reflect on the nature of that interface. For example, in Kitchin and Dodge (2009) we used this technique in order to chart how the code/spaces of air travel are emergent, relational, contingent, and embodied in nature (rather than deterministic, fixed, universal, and mechanistic), and how these code/spaces are bought into being through the interplay of people and

code. Our empirical material was derived through sustained observation of our own and other people's engagement with the software systems that are used to augment air travel. This consisted of spending significant periods of observation as we negotiated different airports and travel between them in order to chart the diverse means and ways of purchasing tickets, checking in, passing through security, hanging around departure lounges, navigating paths to gates, boarding planes, flying, collecting baggage, passing through customs and immigration, and exiting the airport. In so doing, we took detailed notes of events and transcripts of relevant conversations that we could then analyze and compare. Similarly, Ullman (1997) provides an effective self-autobiography of software development that draws extensively on her observant participation.

The power of envisioning methodologies is to reveal previously unknown spatial relationships and to effectively communicate the structure of complex processes. Mapping coded objects, infrastructures, and processes makes it possible to determine their deployment across space and time. Therefore, such a map reveals how particular locales take on the form of coded space or code/space.

Spatialization is a particular kind of visualization in which a spatial structure is applied to capta where no inherent or obvious structure exists in order to provide a means of inscribing and comprehending such capta (Dodge and Kitchin 2000, Skupin and Fabrikant 2003). Here, information attributes are transformed into a spatial structure through the application of concepts such as hierarchy and proximity (nearness/likeness). In previous research, we have documented and used such techniques extensively to examine the nature of several facets of the unseen Internet infrastructure and the social and semantic structures of online media (Dodge 2008, Dodge and Kitchin 2000, 2001). To date, however, mapping and spatialization has been relatively little used as an academic technique to examine the myriad of other ways in which software is being embedded in everyday life, although map interfaces regularly occur in management system interfaces for monitoring particular applications. We believe that together, mapping and spatialization can be used to reveal important insights into who owns and controls coded objects and infrastructure, who has access to them, and how they are being employed.

An example of the possibilities of using software to creatively map out aspects of the work of code, is an online mapping service to search and view different parts of the world. These interactions also leave new kinds of traces of their presence in the world. A pattern memory of their creation is preserved in automatically generated logs of the executing code. These logs can themselves be rendered visually, as maps revealing when and where people are mapping their worlds (see figure 11.1).

Another set of mapping techniques, employed to understand time geographies, also have the potential to reveal when and where people enroll elements of software to

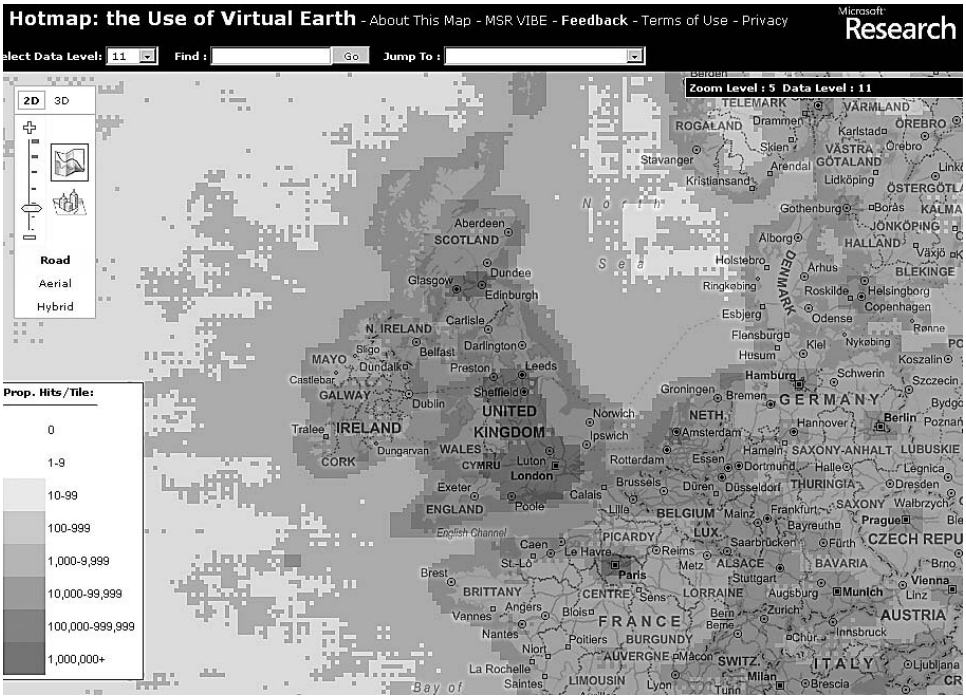

Figure 11.1

The potential to visualize how a mapping algorithm is employed using a simple heatmap approach developed by Fisher (2007). The intensity of colors represents the differential interest levels of users of Microsoft's Virtual Earth mapping system, http://hotmap.msresearch.us/

solve everyday problems. For example, research by a small cadre of human geographers analyzes how individuals rely on computers and online media in relation to the ordering of resources in time and physical/virtual access to spaces (see figure 11.2; Adams 2000; Ellegård and Vilhelmson 2004; Schwanen and Kwan 2008), although they typically do not go into sufficient detail to describe what particular algorithms and capta are at work and the exact nature of the transduction they cause.

These four methods are a subset of the wide range of possible ways of providing the empirical material to underpin theoretical explorations of software studies. As the field develops and grows, no doubt certain conventions will develop as to how research within the field should best be conducted. We would caution against the creation of such hegemonic formations. Instead, software studies needs to be open to a plurality of approaches and techniques, striving to use those tools that provide us with useful empirical material for making sense of the sociality and spatiality of code.

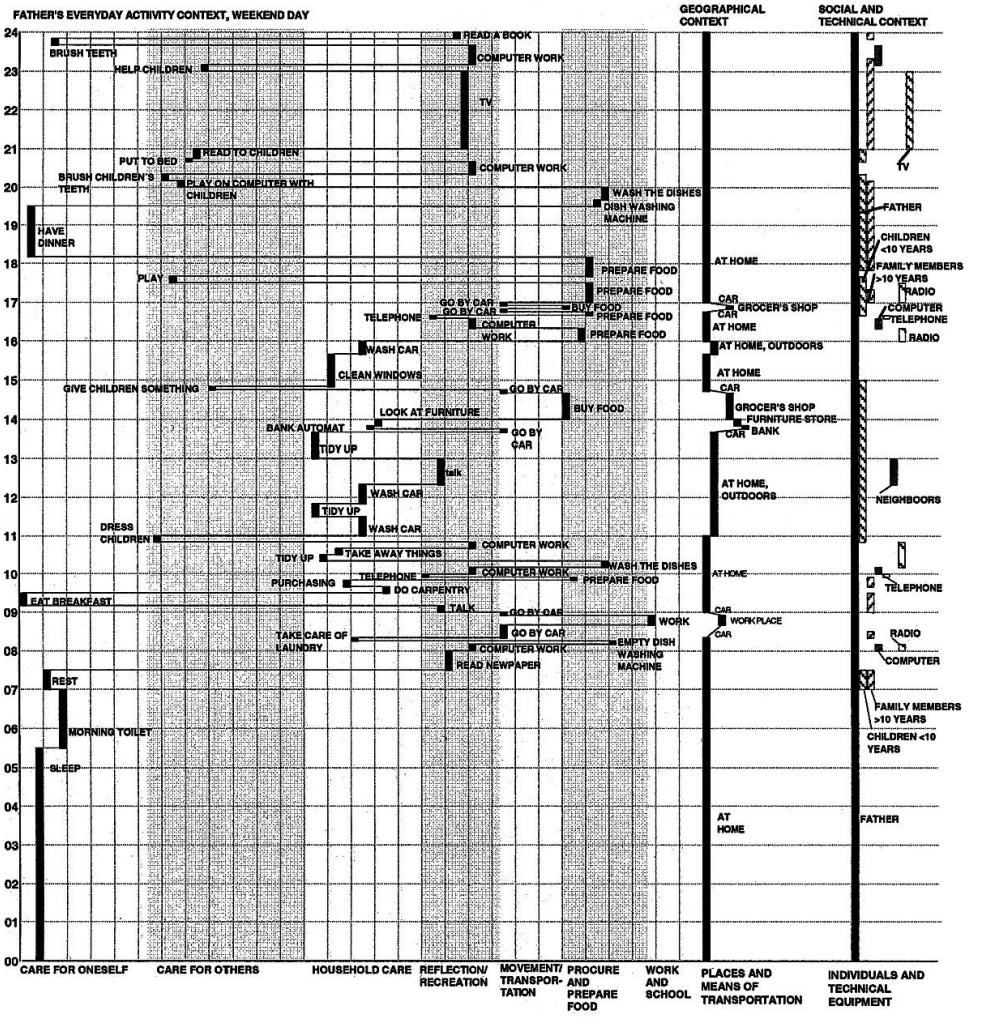

Figure 11.2

A linear visualization of ordering of daily activities through time including indications of when different ICTs are active. *Source*: Ellegård and Vilhelmson 2004, 286

Conclusion

Such has been the rapid growth of software-enabled technologies it is fair to say that code now conditions existence in the West—code is routinely embedded into everyday objects, infrastructures, and systems. Such is its pervasiveness that we would argue it is impossible to now live outside of its orbit (even if one does not directly interact with software, much code is engaged, at a distance, in the provision of contemporary living). As a consequence, the nature of software and its work in the world is urgently in need of serious and sustained intellectual attention from a critical social scientific perspective. This is not to denigrate the work of other scientists studying software from a variety of different vantage points, such as computer science, HCI, or artistic interpretation/critique. These studies and projects often reveal significant details about particular aspects of software, and are important in shaping how future code is produced. Rather, the call for a social science perspective comes from the observation that those within the software industry tend to conceive of code in a narrow, technical, or artifactual sense, instead of as complex and sociocultural productions that do diverse work in the world. Further, this proposed study requires analysis to be concentrated on software itself and not simply the first-order impacts of software-enabled technology. All too often, studies focus on such technologies, ignoring the critical role played by code in shaping their technicity and unfolding solutions to problems. Our aim in this final, brief chapter has been to set out a manifesto for the kind of intellectual endeavor we believe necessary. At the heart of this manifesto is a simple observation—*software matters*. The theoretical concepts and empirical methods for understanding how and why it matters need to reflect its importance in how everyday life comes into being in the twenty-first century.