

ARTICLE INFORMATION

Author

Robert W.Gehl & Sarah A. Bell

Affiliation


University of Utah

Publication Date

28 September 2012

PRINT OPTIONS

 Print This Article

 Download article as PDF

HETEROGENEOUS SOFTWARE ENGINEERING: GARMISCH 1968, MICROSOFT VISTA, AND A METHODOLOGY FOR SOFTWARE STUDIES

The foreword to MIT's Software Studies series suggests its purpose is to "make critical, historical, and experimental accounts of (and interventions via) the objects and processes of software."¹ Methodologically diverse, the international field of software studies welcomes perspectives from the arts and humanities, the social sciences, as well as computer science and engineering. Recent proposals include Noah Wardrip-Fruin's operational logics of expressive processing,² Matthew Kirschenbaum's forensics,³ and Lev Manovich's genealogy of cultural software.⁴ The objective is not to homogenize the study of software, but rather to explore transdisciplinary avenues toward developing "reflexive thinking about [the] entanglements and possibilities"⁵ of computing. Moreover, the field is developing a normative stance towards the powerful organizations behind the software that shapes so much daily life. As Matthew Fuller asks, "What kind of critical and inventive thinking is required to take the various movements in software forward into those areas which are necessary if software oligopolies are to be undermined?"⁶ Indeed, given the power of firms such as Microsoft, Google, Apple, and Facebook, "it is essential for our political future that people develop the ability to think critically about software systems."⁷ Software studies, then, is about radical critique of software systems; it proposes not to take computers "at interface value"⁸ but rather delve into the hierarchical layers of abstraction that comprise hardware and software systems, decompose them, hack them, and alter them in progressive ways.

This essay describes one methodological scaffolding that can be employed in such a cause. Based in John Law's concept of heterogeneous engineering, and taking care to accommodate Kirschenbaum's challenge to software studies for materialist methods,⁹ our proposed methodology and analysis offers an entry point, a conjuncture, at which articulations between product, producer, process, and user can be usefully interrogated.

As Kirschenbaum argues, "Software studies is what media theory becomes after the bubble bursts."¹⁰ That is, this field draws attention to the ways in which digital objects are "interpenetrated by material patterns and circumstances."¹¹ Intangible objects like software are material in that they are the products of processes that include "white papers, engineering specs, marketing reports, conversations and collaborations, intuitive insights and professionalized expertise, venture capital (in other words, money), late nights (in other words, labor), Mountain Dew and espresso.... material circumstances that leave material traces."¹² In sum, Kirschenbaum advocates "a commitment to meticulous documentary research to recover and stabilize the material traces of new media – a remembrance of things past, but also the pre-condition for . . . a 'theory of the present.'"¹³ In other words, such methods promise to uncover the situatedness of intangible objects within the social zones of material and ideological circumstance. To expand on Kirschenbaum's own analogy though, methods and methodologies of software studies research, at least those emerging from a critical vein, must be about more than "adding memory." Adrian Mackenzie acknowledges that "the characteristics of software as a material object, as a means of production, as a human-technical hybrid, as medium of communication, as terrain of political-economic contestation" make software difficult to represent. Its mutable, contingent nature presents both a challenge to research, and makes it imperative that research proceed to interrogate software's "sociality."¹⁴

It is our intention to meet these critical challenges. In keeping with the methodology we propose, this paper is a series of associations. First, we explore the theory of heterogeneous engineering, drawing on the work of Science and Technology Studies (STS) theorists such as Law and Bruno Latour. Next, we read a key document in the

ISSUE ARTICLES

Not just another database: the transactions that enact young offenders

The Algorithmization of the Hyperlink

Relational and Non-Relational Models in the Entextualization of Bureaucracy

Objects of Intense Feeling: The Case of the Twitter API

Critical Codes – from forkbomb to brainfuck

Review of "Inventing the Medium. Principles of Interaction Design as a Cultural Practice"

OTHER JOURNAL CONTENT

Abstract

Reviews

Special Projects

FIND ARTICLES BY AUTHOR

Search for

Find

history of software engineering, the report from the 1968 Garmisch NATO conference, arguing that the engineering metaphor proposed is an association of technological and discursive elements – in sum, we argue software engineering is heterogeneous engineering and should be interpreted as such. After establishing software engineering as a form of heterogeneous engineering, the central case study of the paper is the production of the “Windows Vista Experience,” or the various graphical interfaces included with that Microsoft OS. Using internal Microsoft emails, technical documents, news reports, court testimony, and Microsoft fan writings, we show that the production of a graphical interface does not simply involve the coding of alpha transparencies, smooth window transitions, and “eye candy;” it is an association of heterogeneous elements, including theories of human-computer interaction, graphics cards, chipsets, marketing plans, manufacturers, retail sales clerks, and end users, all of which is intended to cohere into a functioning (and in this case profitable) network. And, as we show, such networks are always unstable, and can be dissociated by material and discursive contestation. Finally, in the conclusion, we will re-link our analysis to the larger field of software studies.

1. Heterogeneous engineering

As Science and Technology Studies (STS) theorists have shown, technological artifacts are not simply neutral tools used by people, nor are they the result of creative geniuses who discover them and bring them to an awaiting world. Rather, artifacts are caught up in webs of discursive and material determinants: discourses, political economic systems, the desires and anxieties of various social groups, the general intellect, and whatever material properties can be brought into the conceptual grasp of the general intellect.

To articulate these webs, STS theorist and sociologist John Law argues that we should start with the “metaphor of the heterogeneous network.”¹⁵ Arising out of Latour and Callon’s actor-network theory,¹⁶ Law’s heterogeneous network “is a way of suggesting that society, organizations, agents, and machines are all effects generated in patterned networks of diverse (not simply human) materials.”¹⁷ Law wants to break discursive and technological totalities into their constituent elements, or heterogeneous elements, to discover “the complexity and contingency of the ways in which these elements interrelate” and document “the way in which solutions are forged in situations of conflict.”¹⁸

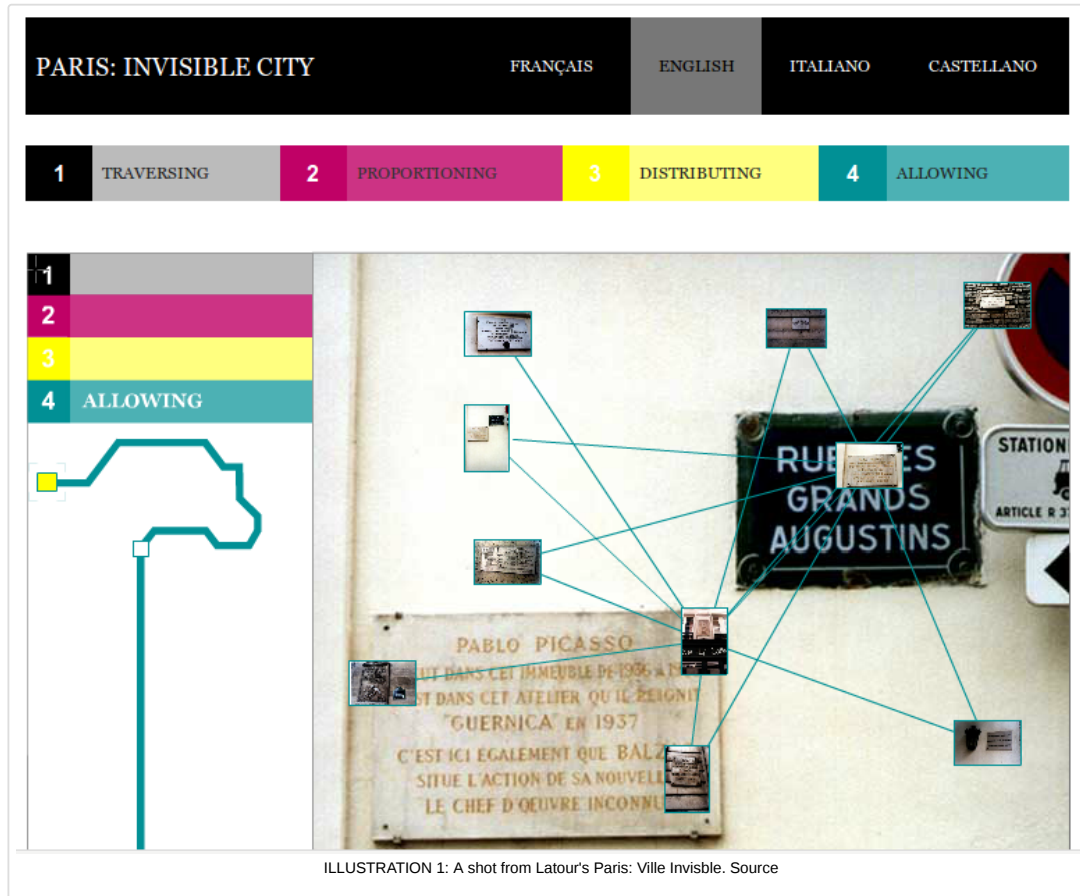
In doing so, we can trace the process of “heterogeneous engineering,” which associates these elements into a coherent system. Krige’s discussion of heterogeneous engineering in science is a concise explanation. This theory

“insists that scientific success demands not just technological innovation with refractory natural materials, but also the mobilization of obdurate human and material resources, the building of alliances, the deflection or silencing of opposition, and the development of persuasive rhetorics. Scientific achievement is a social accomplishment.”¹⁹

In other words, “The argument is that those who build artifacts do not concern themselves with artifacts alone, but must also consider the way in which the artifacts relate to social, economic, political, and scientific factors. All of these factors are interrelated, and all are potentially malleable.”²⁰ Heterogeneous engineering is the building of networks and systems of both technical and discursive elements.

For example, if we consider an object such as a 16th century Portuguese galley ship,²¹ we might simply view it as a successful system: obdurate, powerful, seemingly unassailable, built out of human knowledge, sweat, and not a little imperial ambition. The ship, we might say, is a solution to an apparent problem: the problem of oceangoing trade, travel, and conquest. However, not all is as it appears. “Solutions” in heterogeneous engineering are not simply the best possible technology or technique for the job at hand. Rather, they are “associations” of technological and discursive elements, nearly all of which constantly threaten to dissociate the object in question. The associated elements are “difficult to tame or difficult to hold in place. Vigilance and surveillance have to be maintained, or else the elements will fall out of line and the network will start to crumble.”²² Returning to the example of the ships, all of these elements must be associated and negotiated with: the wood, pitch, cloth, and rope; the

sailors' knowledge, skills, beliefs, and willingness to labor aboard the ship; the Portuguese government's ability to finance ships and desire to put them to sea; the scientific and religious knowledge of the day; myths and legends of sailing; natural phenomena such as wind and waves; knowledge of and cooperation of the stars shining above; geographical possibilities; and technological advancements such as the compass and sailing techniques. All of which crystallizes into a ship, and most importantly, all of which can tear apart a ship or be torn apart by any number of factors: storms at sea, other nations' ships, the failure of the crew, the indifference of the government, technological shortcomings, or simply a bad job building the hull. In short, every heterogeneous element is resistant to being put to the intended purpose.



Thus, the theory of heterogeneous engineering sees – indeed, privileges – two key processes in a technological system: association and dissociation. Both are explained in Latour's *Assembling the Social* and put into practice in his Web project, *Paris: Ville Invisible*. Latour argues that the tracing of linkages between heterogeneous elements is the social, which is to say these linkages are associations. In actor-network theory, the actors (human, nonhuman, material, and immaterial elements) make connections among themselves to form a system. This linking is association, and it “is visible only by the traces it leaves (under trials) when a new association is being produced between elements which themselves are in no way ‘social.’”²³ It is the sociologist's job to faithfully trace these linkages to discover associations. Key here are the ideas of the trial and newness: heterogeneous engineering theory holds that associated elements are always under stress, always changing, and of course, possibly subject to their destruction – i.e., dissociation. That is, associated elements might be pulled apart, dissociated, by internal or external actors which link them together in yet new systems.

In this sense, then, this heterogeneous engineering is sympathetic to articulation theory and Marxian dialectics, which also trace and historicize controversies, contradictions, and

antagonisms and describe how actors stabilize these problems into coherent systems, whether they be technical, discursive, or economic, as well as how those systems can dissociate. More importantly for our purposes here, heterogeneous engineering helps us continue projects such as that of Adrian MacKenzie's *Cutting Code*, which "treats software as a multi-sited associative or concatenated entity."²⁴ Moreover, it also orients us to the tricky question of agency.

The "Heterogeneous Engineer"

The agent that attempts to associate these elements is the "heterogeneous engineer," who is, of course, an effect of a network of heterogeneous elements. Like any expert, the engineer's knowledge is an instantiation of power/knowledge, formulated via access to technological systems and in discursive artifacts like diplomas and certifications.²⁵ The engineer operates from a privileged subject-position. As Krige argues in his analysis of the 1984 Nobel Prize in physics, "only a few, generally outstanding, but always well-connected individuals, situated at appropriate nodes in the social network, are able to weave together the scientific, technological, institutional, and political threads that together constitute their project."²⁶ While Krige does not critically consider precisely why some individuals become "well-connected and outstanding," if we recall actor-network theory, we see that the engineer is a particular effect (as Law might say, a "punctualization")²⁷ of a larger technocratic knowledge and power system. As such, the privileged actor-network "engineer" is more capable of associating elements into a coherent system precisely because this subject is itself associated to do this very task.

To be certain, a theory that holds that some subjects alone are privileged to conquer natural, technical, and discursive systems will come under critique by those who are leery of theories that maintain or reify power inequalities or hierarchies. Susan Leigh Star's work explores this element of heterogeneous engineering. "By experience and by affinity," Star notes,

"some of us begin not with [the manager or the executive], but with the monster, the outcast. Our multiplicity has not been the multiple personality of the executive, but that of the abused child, the half-breed. We are the ones who have done the invisible work of creating a unity of action in the face of multiplicity of selves, as well as, and at the same time, the invisible work of lending unity to the face of the torturer and executive. We have usually been the delegated to, the disciplined."²⁸

Star's emphasis on those subjects who are excluded by sociotechnical standards and practices reminds us to mind the margins and the "monsters" and Others that inhabit them, because of course no system can operate without the labor and affect of the marginalized and disciplined. To only theorize the engineer is to shortchange or even ignore these Others (and, as we will see, Others can find ways to assert themselves).

However, rather than discard the metaphor of the "heterogeneous engineer" because it privileges one particular subject over the multiplicity of others that might associate elements into a coherent system, we want to keep the concept because of this discursive privileging. The concept of "the engineer" – with all its power/knowledge implications – is especially useful to critique a field such as software engineering, because software engineering explicitly uses the metaphor of the engineer as its core organizing principle, as we will discuss below. And, as we will see, software engineering has been defined as such in part against its Others, including other practices of software production, other fields of engineering, the laborers who produce software, and those who buy it and use it. The theory of heterogeneous engineering, as applied to the practice of software engineering, is simultaneously a theory of how engineers do their work and how their work can dominate others (particularly laborers and users). It is thus a theory of power, one very capable of allowing us to critique the development, uses, heteroclines,²⁹ and lacunae of software artifacts. Thus, we can hold onto the concept of the engineer, so long as we mind its margins and the Others that help constitute it or even overwhelm and destroy it.

2. From "mushyware to firmware": the software engineering metaphor

Methodologically, of course, applying this theory of heterogeneous engineering is incredibly complex. As Latour notes, "one has to substitute the painful and costly

longhand of... associations" for the quick and easy abstractions we are used to.³⁰ If technical and discursive phenomena are given the same weight, if we are to move past simple cause-and-effect to overdetermination, and if we are to trace articulations to see how elements are associated into a coherent system (one which could dissociate at any moment!), where do we begin, especially since we are looking at software? Do we start with lines of code, algorithms, the platform(s), the habits of programmers, the desires of users, the marketing of software, the global ambitions of transnational software company CEOs? Fortunately, the metaphor of engineering is not only deployed in heterogeneous engineering; it is deployed in software production as well. This metaphor provides us with a few entry points.



ILLUSTRATION TWO. A panel of participants in the 1968 Garmisch conference on Software Engineering. Last names, from left to right: Smith, Paul, Perlis, Randell, Ross, Graham, Goos, van der Poel, McIlroy, and Kinslow. Source: <http://homepages.cs.ncl.ac.uk/brian.randell/NATO/N1968/GROUP7.html>

As Mahoney notes, the electronic digital stored-program computer was a the product of a convergence of two nineteenth century developments, the design of mechanical calculators and the development of mathematical logic. Both viewed programming as incidental, but as the computer went commercial in the 1950s, the need for quality programming became of sudden concern, and those stepping in to fill the need for programmers were often not scientists, mathematicians, or electrical engineers.³¹ Prompted by recognized "crises" in software production, including a shortage of programmers, poor reliability of many programs, and unsustainable cost overruns,³² in 1968 the NATO Science Committee convened a conference in Garmisch, Germany under the title "Software Engineering."³³ "The phrase 'software engineering' was deliberately chosen as being provocative, in implying the need for software manufacture to be based on the types of theoretical foundations and practical disciplines, that are traditional in the established branches of engineering."³⁴ The academics, managers, programmers, and marketers who participated in the conference adopted the discursive and technical concept of "engineering," thus associating the practices of software production with respected and established practices such as building bridges and

buildings. Related metaphors such as "architecture," "implementation," "specification," and "large systems" are also debated, deployed, and developed by the participants throughout the conference proceedings. To be certain, when taken literally to mean a shared body of professional practice based in the mechanics of the physical world, "engineering" applied to the production of software remains controversial.³⁵ However, the metaphor of engineering persists to this day as evidenced by disciplinary documents such as IEEE's Guide to the Software Engineering Body of Knowledge, the ACM's Software Engineering Code of Ethics and Professional Practice, and recognition of ABET as the accreditation body for computer science and software engineering programs in the U.S..

Of course, because the engineering metaphor was only newly proposed for software production, it was in need of much elaboration during the conference. Since computer hardware production was based on more established practices such as electrical and metallurgical engineering, the participants in the NATO conference often pointed to hardware production as a discursive and methodological model for software production. In this comparison, however, software people felt inadequate. M.D. McIlroy noted, "We undoubtedly get the short end of the stick in confrontations with hardware people because they are the industrialists and we are the crofters. Software production today appears in the scale of industrialization somewhere below the more backward construction industries."³⁶ As A. d'Agapeyeff argued, "Programming is still too much of an artistic endeavour. We need a more substantial basis to be taught and monitored in practice on the: i) structure of programs and the flow of their execution; ii) shaping of modules and an environment for their testing; iii) simulation of run time conditions."³⁷ J.W. Smith argued, "There is a tendency that designers use fuzzy terms, like 'elegant' or 'powerful' or 'flexible'.... What is lacking is discipline, which is caused by people falling back on fuzzy concepts, instead of on some razors of Occam, which they can really use to make design decisions."³⁸ In his keynote, A.J. Perlis summed up the problem: "Our goal this week is the conversion of mushyware to firmware, to transmute our products from Jello to crystals."³⁹ The engineering metaphor promised to provide the language, rigor, and methodology to shift software production from a "backwards" or (perhaps worse) "fuzzy," "artistic," or "mushy" field to one that could hold its own against other forms of engineering.

In addition, by adopting this metaphor, software producers could draw on all the related practices involved in engineering, particularly those of managing labor. Returning to d'Agapeyeff's lament, if software production is an art, then managing software producers (who would be, following the metaphor, creative artists) is a very difficult task. However, by drawing on the engineering metaphor, participants in the conference began the process of establishing workflows, decomposing tasks into small, manageable parts, predicting costs for each stage, and organizing programmers into hierarchies.⁴⁰ M.D. McIlroy advocated "mass production techniques" including subassemblies and reusable modules.⁴¹ Bell Lab's J.A. Harr proposed various quanta for managing labor: number of programmers, years, "man years," words (i.e., a fixed number of bits), and words per man year. The literature following the 1968 conference is replete with references to these quanta.⁴² These management techniques are part and parcel of the engineering metaphor, and they allow software companies to discipline their workforce. Returning to Perlis's keynote, "Stability in our goals, products and performances can only be achieved when accompanied by a sufficient supply of workers who are properly trained, motivated, and interchangeable."⁴³ The engineering metaphor allows software companies to imagine (and discipline) workers who are not artists or researchers but rather engineers working in a strict hierarchy with measurable goals and outcomes. And of course, if any worker is not producing the required words per man-month, he or she is interchangeable with another, more productive worker.

Finally, the use of the software engineering metaphor altered the relationship between producers and consumers. Rather than programming as an art form, or programming for research and experimentation, the engineering metaphor implies programming for use. This of course orients production towards particular, idealized forms of consumption: unlike artistic production which is meant to be appreciated for aesthetic purposes, or research production which is meant for producing knowledge for knowledge's sake, engineering produces for specified tasks and purposes. Engineering also delimited what a software producer should promise to a user: the adoption of the term "specification"

allowed software engineers to think of their task as the production of objects that do specific tasks and no more, thus heading off unattainable user expectations.⁴⁴ While users are the target for software production they also have to be managed, because "The customer often does not know what he needs, and is sometimes cut off from knowing what is or what might be available."⁴⁵ User demands have to be filtered and mediated by those on the inside.⁴⁶ Following Woolgar, this process might be called "configuring the user," constructing the user as an entity "outside" the software production firm whose expectations and practices have to be carefully managed.⁴⁷

Historians of software now agree that the 1968 NATO conference in Garmisch was the landmark point at which "engineering" became the predominant paradigm for the development of software in both industry and academia. It was the start of thinking about the organizing methods and disciplined practices that were needed to develop increasingly large-scale software products.⁴⁸ Beyond the conference, the engineering metaphor has persisted in software production. Organizations such as the IEEE, ACM, and ABET have solidified the metaphor in technical manuals, training programs, academic and industry conferences, and the production of industry standards.

Indeed, the engineering metaphor has persisted through tumultuous changes in the history of software, including the unbundling of software from hardware and the transition from mainframes to minicomputers to personal computers. Boehm characterizes the history of software engineering as one dominated by reactions to scalability. The engineering metaphor has been adapted through the 1980s focus on productivity, the 1990s focus on concurrent processes, and the more recent trends towards agility and global integration.⁴⁹ Since its rise to market dominance in the personal computer software industry, Microsoft has explicitly adopted models of software engineering, using the category to describe not one role but the "collection of disciplines responsible for designing, developing, and delivering" its products.⁵⁰ Its engineers are defined as "[thriving] on simplifying people's jobs and lives and helping them reach new heights through ever-advancing technologies."⁵¹ Judging from popular reports on Microsoft labor practices, such engineers reach for such new heights within a highly hierarchical structure of decomposed tasks and long hours obsessing over bug reports.⁵² Implying both the management hierarchy and user focus of the software engineering paradigm, Microsoft further articulates this commitment in publications from its research arm, Research in Software Engineering (RISE).⁵³ For example, in a recent paper presented at the ACM Conference on Computer Supported Cooperative Work (CSCW), Microsoft researchers discussed their empirical methods, specifically invoking Brooks' discussion of workflow coordination in *The Mythical Man Month* as a touchstone for discussing the "developer-module network" for Windows Vista, a network that takes into account "thousands of developers and thousands of binaries – executables (.exe), shared libraries (.dll) and drivers (.sys)."⁵⁴ Their conclusions echo Brooks' declaration that product quality is strongly affected by organizational structure. Indeed, in a well documented shake-up during the development of what became the Vista OS, Jim Allchin, co-head of the Platform Products and Services Division, and Amitabh Srivastava were tasked with streamlining processes to overcome a dysfunctional environment of 4000 programmers and testers working with 50 million lines of code.⁵⁵

Immediately, it should be apparent that software engineering from Garmisch 1968 through today is heterogeneous engineering par excellence, because the twin processes of association and dissociation are clearly present. Software engineers associate technological elements (computer platforms, hardware, programming languages, user interfaces), human elements (managers, coders, users, education, corporations), and discursive elements (metaphors, sales pitches, theories of needs) into cohesive artifacts. And, in keeping with the concept of the engineer as an effect of a larger, rationalized power/knowledge system, software's use of the engineering metaphor reminds us of the highly hierarchical nature of this form of production. As Garmisch conference participant Peter Naur argues, "... software designers are in a similar position to architects and civil engineers, particularly those concerned with the design of large heterogeneous constructions, such as towns and industrial plants. It therefore seems natural that we should turn to these subjects for ideas about how to attack the design problem."⁵⁶ That is, the engineering metaphor can provide software producers with the power/knowledge to deal with large problems in the same privileged manner as civil engineers have enjoyed in modernity, including the organization of labor and the conceptualization of end

use. We can see this in Crutzen and Kotkamp's more recent, concise, and useful definition of software engineering: "The goal of software engineering is to produce unambiguous software that masters complexity. Based on the principles of controlling complexity and the reducing ambiguity within software, software engineers try to tame the complexity and ambiguity of the real world."⁵⁷ Thus, the metaphor of "engineering" provides us with some key starting points for analysis: hierarchies (both in layers of abstraction in software and in the organization of labor), the relationship between hardware and software, and the inside/outside dichotomy of producers and consumers.

And yet, we cannot simply take the engineers at their word. Heterogeneity enters into our analysis when we consider how every element the engineer works on or around may also work in reverse upon the engineer, or when unexpected elements disturb the carefully built associations. One is reminded of the first computer bug, the moth Grace Hopper pulled out of the Harvard Mark II. Moths short-circuiting relays are only the most salient elements in a heterogeneous and often unpredictable stew comprised of electrical currents, humidity, dust, electrons, temperature,⁵⁸ metals and plastics, chipsets, assemblers, programming languages, programmers, garages, converted wool mills, state tax incentives, universities, textbooks, lines of codes, documentation, global supply chains, endless loops, interpreters, California hippies, emulators, MIT hackers, managers, divorce lawyers and psychiatric wards,⁵⁹ Irish entrepreneurs, New York investors, Indian tech workers, end users, Chinese Shanzai copycats, licenses, and hardware configurations. Software's relationship to the future and its surprising obduracy also can undermine the engineer's efforts at association; the Y2K Bug panic occurred because software systems lasted long enough that it mattered that variables for years were encoded in two decimal digits and not four. The history of software is replete with elements such as these irrupting into the smooth workings of would-be master software engineers. For all the idealism of the engineering metaphor's ability to help tame this complexity, the anxiety of Garmisch 1968 and software engineering in general arises because many times software engineering simply means working on contingency, accident, failure, and unexpected success.

Thus, it should be clear that the cohesive artifact called "software" is always under threat of dissociation by the unruly elements of the "real world," however well those elements are fended off. Indeed, the history of software production is full of stories of overpromises and underdeliveries, missed deadlines, and failures.⁶⁰ IBM's System/360 is a famous example. ALGOL's failure to become an international standard high-level, cross-platform (read: not dominated by IBM) language is another. The demise of Lotus 1-2-3 points to another dissociation, this time due to Lotus's engineers betting on IBM's OS/2 operating system when Microsoft Windows was about to ascend to dominate the PC market. A more recent example is the failure of MySpace.⁶¹ However, rather than go on listing examples, the next section will explore in detail the heterogeneous engineering of Microsoft's Windows Vista and its subsequent dissociation in order to demonstrate the use of heterogeneous engineering as a mode of critical and normative inquiry for software studies.

3. Windows Vista

Windows XP was a massive success for Microsoft, and its successor, "Windows Vista," was highly anticipated. However, today many consider Vista to be a failed software system. How was it dissociated? Since Vista is a complex system, there are many areas to explore: securing (and securitizing) the operating system with User Access Control, Digital Rights Management, attempting to limit pirated distribution of Vista, "Trusted Computing," and limiting third-party access to the kernel via signed driver files; the advent of 64-bit processing; Microsoft's run-ins with U.S. and E.U. anti-trust regulators; the dominance of Windows and Microsoft in gaming; Vista's relationship to Mac OSX, Linux, and the emerging cloud-based software of companies like Google; or (importantly) the day-to-day life of wage workers at Microsoft as they worked long hours to code and debug Vista. Here we will explore one case: the heterogeneous engineering of the graphical interface, Aero, the production of the "Vista Experience," and how it was bifurcated to appease part of Microsoft's network, Original Equipment Manufacturers (OEMs) and Intel, to the detriment of another, end users.

Engineering Aero and the "Vista Experience"



A large collection of heterogeneous bits and pieces went into construction of the discursive and technical system referred to as the "Vista Experience." Over the period of time when Vista was produced (roughly 2003 to late 2006), Microsoft management worked to associate external elements such as processor manufacturers (notably Intel), "original equipment manufacturers" (OEMs, notably Dell, Hewlett-Packard, Sony, and Gateway), hardware manufacturers (makers of scanners, cameras, printers, and other peripherals), chip makers (Intel, AMD), retailers (Best Buy, Dell), and an object alternately called The Customer, The Masses, and The Consumer.⁶² Moreover, Microsoft itself had to associate its own internal elements (2,400 developers, 2,500 testers, and 1,500 program managers,⁶³ marketers, accountants, internal computer systems) to align with this external network.

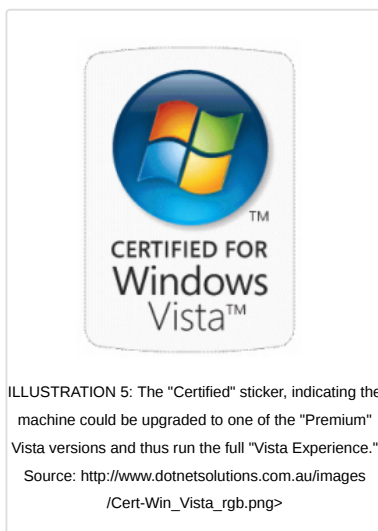
This complex process was aimed at producing a software system repeatedly referred to as the "Vista Experience." Given that windows-based graphical interfaces are the established surfaces with which users interact with their machines, the "Vista Experience" first appears to be a new graphical system, Windows Aero.⁶⁴ Although it is the highest layer of abstraction in a personal computer (and thus it is only one small part of the computer system), for most users, the graphical interface is the computer. When fully utilized, Aero provides 3-dimensional "flip" navigation through open windows, semi-transparent window decorations, previews of open programs, and sleek menus. To see how users react to such "eye candy," simply search YouTube for videos of Windows Aero.⁶⁵ This emphasis on images, visual culture, and graphics ("windows," "vista," etc.) is a longstanding one for Microsoft, starting with the grandiose launch of Windows 95 to a Vista marketing stunt in which aerial dancers formed the Windows logo on the side of a building in New York.⁶⁶

However, we cannot remain at the surface, at the visual sign. We have to see Microsoft's Aero as an attempt to associate multiple heterogeneous elements: new (and anticipated) processors and graphics cards that could render 3D images; wide aspect ratio high-density LCD monitors; users who are attracted to OSes with "eye candy" (i.e., Mac OS X, Linux with compositing); previous Windows users who are fans of Microsoft's software; computer industry critics who look for new technologies to write about in magazines and on blogs; the visual signs of market share, as well as shareholder perception of the firm; retailers in need of new commodities to sell; and the larger image-conscious visual culture of marketing and postmodern computing.

This association of hardware and software was aimed towards creating a Vista Experience that is emotional; as Microsoft critic Paul Thurrott notes, "Like its predecessors, Windows Vista is ultimately just a tool, of course, but it's one that provides you with an emotional relationship that you just didn't get from Windows XP. Those new qualities extend from the packaging of the system, to the new system sounds, to the new visual capabilities and glass-like icons, to the subtle animations, and other features."⁶⁷ The "Vista Experience" thus becomes a central, organizing mythology for all of these elements to be arranged around. Thus, the engineering of the Vista experience is the engineering of affect and technology, graphics and hardware, screens and machine-level events. For example, early in the production process (in 2003), Will Poole, who was a Microsoft Senior Vice President during the Vista development, described the emphasis on "immersive" graphics as a reaction to the visual and technological experiences offered by video games.⁶⁸

However, despite the emphasis on Aero and technical engineering of emotional graphics, Microsoft could not demonstrate this graphical "Vista Experience" too early; in fact, it had to hold it in abeyance. It did so because two vital parts of its associated network, OEMs and retailers, did not want sagging sales of computers with Windows XP in the months prior to the release of Vista in January 2007. Their collective theory was, if consumers saw Vista and knew it was coming in the next year or so, they would hold off buying computers until the new OS was released.⁶⁹ Thus, although Microsoft was intently building the Vista software system and touting its graphical innovations at trade shows and to tech reporters, it faced a threat of dissociation from OEMs and retailers: if Microsoft undermined OEM and retail sales by promoting Vista too soon, Microsoft would run the risk of losing influence with those elements of the network. Retailers are a special concern, because the larger ones (such as Best Buy) already sell Apple products and could increase their emphasis on Apple at the expense of Microsoft. And of course, Microsoft itself would be hurt by slow sales of Windows XP-loaded computers, and another element in its network, shareholders, would threaten the overall system.

Complicating the surveillance of the OEM and retail networks, Microsoft had to maintain association with another heterogeneous element: Intel. While OEMs such as HP and Sony make computers, they don't make the required microprocessors. Intel's dominance here requires both software and hardware firms to associate new chipsets into their architectures, as well as maintaining sales of older chipsets. In the case of Vista, the early (2005) requirements of the Aero graphical interface (the Windows Display Driver Model) were high⁷⁰ – so much so that the vast majority of chipsets on the market, including Intel's 915, could not meet them. At that time, Intel was introducing the 945 chipset, which could provide the power to run Vista Aero.⁷¹ However, Intel would not be ready with the massive production of the 945 in time for the launch of Vista (then scheduled for 2006, eventually pushed back to 2007); additionally, the 945 was significantly more expensive than the 915.⁷² Instead, the bulk of their production was still tied to the older 915 set. Since the 915 was not capable of meeting the Aero specifications and thus producing the "Vista Experience" Microsoft intended,⁷³ Microsoft management would have to choose: continue with the high requirements for the graphical "Vista Experience" or somehow dilute it to enable Intel – an important part of Microsoft's network – to sell its stock of older 915s; associate an uncertain (and yet, as always in computational culture, a better and faster) future, or retain a strong link with a past element that enabled Microsoft to be the giant corporation it was.



In order to maintain their associations with retailers, OEMs, and Intel, Microsoft management shifted the "Vista Experience" from its on-screen, on-computer, graphical and emotional phenomenon to two non-software, non-hardware phenomena: a sticker and a sales pitch.

The sticker was an attempt to maintain sales of computers with Windows XP ahead of the launch of Vista. Rather than publicly tout Vista's capabilities, Microsoft management

decided to affix stickers to computers that proclaimed the computers were “future-proof”; that is, that they could upgrade to Vista when it was released. Retail sales associates were trained to sell XP computers based on a future promise of an unseen, unknowable, and yet attainable “Vista Experience.” This was intended to aid Microsoft’s networks of OEM and retailers, who could maintain sales even as the future Vista loomed.

However, the most significant change to the future “Vista Experience” arose due to Intel’s desire to sell more 915 chipsets. To maintain the association with Intel, Microsoft marketing decided on two levels, signified by two different stickers: Windows Vista Capable and Certified for Windows Vista (see illustrations 4 and 5). Computers marked with the “Vista Capable” logo would not have the graphics power to run Aero; the version of Vista here is “Vista Home Basic.” In most cases, this meant the computer had an Intel 915 chipset. The experience anyone with a “Vista Capable” 915 machine would have is not, then, the idealized “Vista Experience” described above: 3D graphics, flip navigation, transparencies, and the like. But rather than admit this is not an “experience,” Microsoft marketers referred to day-to-day use of Vista Basic as the “core experience” of Vista. This includes “Parental Controls, Windows Photo Gallery, Windows Defender, and Instant Search.”⁷⁴ The “Certified for Windows Vista” logo was affixed to computers that met the original Aero requirements – typically Intel 945 machines with Vista Home Premium, Vista Business, Vista Enterprise, or Vista Ultimate editions. With this bifurcation of “experience,” Microsoft was able to associate Intel’s 915 chipset into its network, thus allowing Intel and OEMs to sell more of their older hardware.⁷⁵

This bifurcation between the (premium) “Vista Experience” and the (non-premium) “Core Experience” points to a familiar quandary in modern graphics-based software production: the use of abstractions to hide lower levels of the machine.⁷⁶ The highest level of abstraction in an OS is the graphical interface. Typically, if this layer is complex, it demands much more of the underlying software (especially drivers) and hardware. To purchase a computer capable of running Aero and 3D graphics before the Vista release in 2007, a user had to be familiar with the inner workings of computers, especially the graphics card, the processor, the amount of RAM, and so on. Paradoxically, however, the glossier and smoother this OS layer is, the more “user friendly” the machine appears. From the earliest days of the GUI – and here we’re thinking of Macintoshes especially – a consumer’s expectation is that a computer would “just work”; that is, the user would not need knowledge of the inner workings of the machine, and would instead simply use GUIs to manipulate the machine.⁷⁷ Thus the GUI – the surface – often stands in for and elides the material machine beneath it.⁷⁸

Thus, when Aero is demonstrated or discussed, the underlying assumption is that this glossy surface and 3D navigation is in fact Windows Vista. And herein lies a major problem in Microsoft’s production of the “Vista Experience” out of heterogeneity. If Aero is Vista, to be “Vista Capable” seems to mean “Aero Capable.” Aero’s future-deferred and (possibly, depending on the chipset) technologically-prevented visual and aesthetic promises haunted constructions of Vista before its release. At that time, the heterogeneous elements comprising Vista oscillated and mutated depending on context. For a shop-floor sales clerk to explain to “the consumer” what “Aero,” “Parental Controls,” “Windows Defender,” and sundry other details mean – all ahead of the Vista launch and without a Vista-loaded computer – was a difficult matter of configuring a system comprised of promises that oscillate between the ideal and the material, between Microsoft’s successful past and its presumably successful future, between subtle conceptions of “premium” graphics and “capable” graphics, and between a computer in the store with the known OS (XP) and the next-generation OS. For “the consumer,” Vista-in-the-future was an OS capable of improving in largely unknown ways upon the material machine they were purchasing at that moment (the machine loaded with XP). The consumer’s screen, keyboard, mouse, and peripherals would (presumably? hopefully?) alter, extend, and improve their capabilities with a new OS. In this sense, “future-proof” did not mean that a computer purchased today would work tomorrow; future-proof means that the computer has to deliver the better tomorrow always promised (and endlessly deferred) in consumption-based capitalism. And, after the Vista release, retail salespeople had to explain to disgruntled consumers why some computers didn’t run Aero, which was, after all, supposed to be “The Vista Experience” all along. Thus, the material machine asserted itself in myriad ways, depending upon its configuration, altering the promised relationships between Microsoft, retailer, consumer, hardware,

peripherals, software, and the future. This confusion would emerge as a major dissociating force of Vista, a point we will return to later.

Rajesh Srinivasan as Heterogeneous Engineer

So far, this is a high-level view of the heterogeneous engineering of Vista. Here, we want to focus on one Microsoft heterogeneous engineer in particular, Rajesh Srinivasan, who was instrumental in maintaining the associations between Microsoft and OEMs. Srinivasan, a Senior Manager at Microsoft since 2004 who has degrees in engineering and management, is positioned high in the management hierarchy, answering to vice-presidents (such as Mike Sievert and former VP Will Poole), the Chief Technical Officer Ray Ozzie, co-President Jim Allchin, and the CEO Steve Ballmer. Based on internal Microsoft emails (made public by court order in *Kelley v. Microsoft*), we can show how Srinivasan negotiated with heterogeneous elements in order to build and maintain the Vista heterogeneous network.

First, because Intel's 945 chipset was more expensive and available in smaller quantities, Srinivasan had to contrast the idealized Vista Aero experience with the day-to-day reality of OEMs, retailers, and Intel. In a key August 2005 email thread to Keith Eagen and Mike Croft, Srinivasan argued for a two-tiered approach to producing Vista:

"Although a single tier program positions things as binary – PC is either Ready or Not... By using a two tier approach, We [sic] can still launch the program in Jan 06 at CES and maximize our up-sell opportunity. By letting OEMs know now, we have a small chance of influencing their product plans for Spring and definitely for Summer. We are not relaxing any criteria [in terms of graphics] that impacts user experience on PCs that are labeled Ready. And we avoid customer dissatisfaction. Although this sounds like a win-win, we would definitely have some strong OEM push back, and would need your support and the OEM division's support to get OEM buy-in. And we would have to seek Best Buy's help to convince with [sic] their OEMs (HP, Gateway, Toshiba, Sony), which they have tentatively offered today."⁷⁹

Here, Srinivasan attempts to negotiate the needs of OEMs and retailers such as Best Buy with Microsoft's need to "maximize our up-sell opportunity." He argues that the early negotiation of the two-tiered plan may also influence what OEMs produce. He also weighs the full Aero experience ("We are not relaxing any criteria that impacts user experience") against the need to sell more computers now, and the need to sell more computers now against "customer dissatisfaction." Finally, anticipating "strong OEM pushback," he asks higher-level Microsoft management and Best Buy for help in convincing OEMs to adopt the two-tiered plan.

Although this sounds as if Srinivasan is "caving to Intel" (as another Microsoft employee puts it⁸⁰) because the two tiers would center on the 945 and the 915, he sees the two-tiered approach as a chance to use OEMs to discipline the lagging Intel: "This is more about not letting OEMs lock in their product plans fully on 915, then [sic] letting Intel game us."⁸¹ He argued for delaying Vista enough to "use that as leverage with OEMs to put pressure on Intel to end of life 915 by Oct 06, that is a big win for us."⁸² Moreover, Srinivasan makes one last push to save the full "Vista Experience" and to remove Intel's 915 from the network: "Brad [Goldberg] and Shanen [Boettcher] now agree that single tier, Vista Optimized in May/June [of 2006] is optimum. Hopefully with their support and OEM div support, we can convince [VP Mike Sievert] and then [VP Will Poole]."

Of course, this single-tiered plan failed, and the 915 worked its way into the network. However, Srinivasan again worked to maintain (as well as he could) the "Vista Experience" by denying the inclusion of an even older Intel chipset, the 865, as "Vista Capable." In early 2006, Rick Nolte emailed him, noting after Vista dropped the standard for "Capable", they got "slammed from numerous customers in virtually every geography telling us that MSFT has communicated that Intel 865 based platforms qualify for the Vista Capable PC program."⁸³ The email exchanges that follow indicate the ambiguity of this situation – no one could answer definitively that the 865 would not qualify for Vista. Srinivasan fought against this backwards-expansion, arguing that, while it would help Intel sell more chips, it would lead to "customer dissat" with the Vista Experience.

Similarly, in early 2006, Srinivasan had another – decidedly less technical – management challenge when he learned that the OEM Sony wanted to go into retailers and remove old “Designed for Windows XP” stickers and replace them. Sony wanted to do so because “apparently retailers, esp Circuit City and CompUSA are telling OEMs that on April 1 [2006] systems not logo'd Capable would get marked down or returned.”⁸⁴ That is, any computer without one of the new stickers would be discounted heavily, and Sony was afraid they would lose \$20 million.⁸⁵ However, allowing an OEM to replace stickers brought up a sticky issue: Microsoft's stickers use a strong adhesive, and Srinivasan and his colleagues were concerned that the OEMs would do a poor job of cleanly replacing them.⁸⁶ Moreover, they were concerned that the OEMs would mistakenly label computers “Certified for Vista” when they were merely “Vista Capable,” thus putting stickers on the wrong potential experience.⁸⁷ To allay Sony's concerns, Srinivasan decided to deploy another sign, a “retail fact tag,” saying the computers in question would in fact run Vista despite their “Designed for Windows” stickers.

However, despite Srinivasan's efforts at associating OEMs, myriad hardware configurations, retailers, signage, stickers, Intel, other Microsoft employees (marketers, vice-presidents, managers, and engineers), Vista is largely considered to be another failed software product in the long history of software mishaps. The heterogeneity of the “Vista Experience” proved to be too difficult for him or others at Microsoft to tame. The deferred future of slick graphics rendered quickly on the screen oscillated with the existing hardware configurations produced by OEMs and Intel. In the end, as we will explore next, it was the user who bucked the proposed network of associations.

4. Conclusion: Vista's dissociation, and implications for Software Studies

It is tempting for cultural critique to interpret materiality based solely in political economic conditions (i.e., capitalism, racism, or globalization). Likewise, critiques of software often treat it as an immaterial good, and thus as a discourse to unpack or a set of lines of code to parse.⁸⁸ Making materiality and immateriality meet – or better yet, resisting such reductions – is a stated goal of software studies. For example, Mackenzie warns against research which “implicitly pre-fabricates [software] objects as social or cultural” and challenges scholars to “[revisit] concepts of social, cultural, and technological.”⁸⁹ We believe an approach based on heterogeneous engineering can avoid this material/immaterial dichotomy, because heterogeneous engineering requires inquiry to begin within the network, identifying nodes of producer, product, process, and use, tracing flows of influence and determination, all the while being agnostic as to any one heterogeneous object's materiality or immateriality, biology or technicity.⁹⁰ But despite the emphasis on networks, the heterogeneous engineering approach seeks to avoid the “hegemony of network flows”⁹¹ and pre-defined network roles by making the articulations⁹² and associations between nodes the subject of investigation. In order to identify the sociality of software,⁹³ critical scholars must go beyond the dialectical interaction of economic base and cultural superstructure and instead explore the “uneven configuration of different levels of production”⁹⁴ by providing diagrammatic accounts of association and dissociation. After all, digital objects are the product of dynamic, cumulative, and disruptive processes of interpretation, reiteration, citation, and death. Heterogeneous engineering provides a conjuncture for investigation as a “spatio-temporal articulation of different apparatuses forming a diagram of power.”⁹⁵

This can be made clear by a final consideration of the “Vista Experience,” specifically how it can be conceived of as a failed heterogeneous network of associations. As early as August 2005, in an email to Srinivasan, Mark Croft noted that “it TOUGH [sic] to educate consumers” on the two-tiered “Capable and Premium” software plan.⁹⁶ He turned out to be right. Despite their repeated warnings about “customer dissat,” by working to associate OEMs, Intel, and retailers such as Best Buy, Srinivasan and the heterogeneous engineers at Microsoft gave less attention to the end user of the system. This was despite repeated cries from a few Microsoft employees on behalf of the “average consumers.” The exemplar is Brad Goldberg's email from November 2005:

Customers may not have any context from phrases like 'Aero Glass or Windows Defender or Sideshow' ... The average consumer would not know whether (s)he needs Aero-Glass or Windows Defender or not. Retail sales person [sic] cannot explain what Aero Glass is or what it will do for them four – six months prior to Vista launch. ... It takes us

incredibly long time [sic] to explain to OEMs the benefits and value prop[osition] of each feature/scenario. How can we communicate this to an end-user in a document, when vast majority of customers can't understand what an OS does for them?... Trying to 'educate' customers about features of an OS that is not available may well confuse them and may cause them to delay their purchase – the exact opposite of what we want to see. Less than 5% of customers typically upgrade OS. Let's not confuse the masses for the sake of providing clarity to 'enthusiasts'.⁹⁷

And Jim Hebert frankly quipped, "Vista Capable is an XP program – not a Vista program,"⁹⁸ thus noting that the end-user wasn't the concern; rather, maintaining sales of XP was.

These internal emails are available because consumers brought a class-action lawsuit against Microsoft in 2007, alleging that the "Vista Capable" program was intended to inflate the prices of older, XP-loaded machines. Beyond this legal challenge, when Vista was finally released, there was much "customer dissat": InfoWorld started a "Save XP" petition to extend the life of the popular OS because so many users did not like Vista.[93 Gruman, "The Campaign to Save Windows XP."] Microsoft in fact did extend XP's service life for several months in 2008, allowed consumers "downgrade rights" from Vista to XP until 2009, and allowed manufacturers to load it on netbooks (as of this writing, one could still purchase an XP Service Pack 3 netbook new from Amazon). XP remains the number one OS in the world, even ahead of Windows 7.⁹⁹ Despite proclamations from Microsoft that Vista was a success,¹⁰⁰ the "Vista Experience" was ultimately dissociated, because it failed to associate users into the actor-network. Returning to Star's warning,¹⁰¹ the user was meant to be a disciplined outcast in the Microsoft network, happily using whatever OS Microsoft and retailers sold, but thanks to other networks (i.e., America's trigger-happy litigious culture and its cultural emphasis on consumer rights as democratic rights), the users reasserted themselves.¹⁰²

This dissociation – this unmasking¹⁰³ of the normally neat, self-contained software abstraction known as Microsoft Windows – alerts us to the highly contingent nature of software, something that the participants in the 1968 NATO Garmisch conference recognized as they debated and ultimately took up the engineering metaphor. Software's complexity requires software studies to draw on the very metaphors used by its producers, as well as to seek out heteroclitics and oscillations within and beyond those metaphors. Thus we feel that this theory of heterogeneous (software) engineering can be useful to the field. Indeed, considering the arguments in *Software Studies: A Lexicon* and in Bogost and Montford's platform studies,¹⁰⁴ we are not alone in critically thinking about engineering. As Goffey argues, "Locating itself squarely on the side of the reductionist strategies of the exact sciences, society, culture, and politics are very much marginal to the concerns of computing science. Software engineering, on the other hand, concerned as it is with the pragmatic efficacy of building software for particular purposes, might appear to offer a better starting point for factoring culture back into software."¹⁰⁵ We agree, and we hope the theory of heterogeneous software engineering can further this line of analysis.

Finally, on a normative note, we return to Fuller's challenge for us to undermine software oligopolies with critique and praxis.¹⁰⁶ For those of us concerned about the dominance of Microsoft, Apple, Facebook, Cisco, IBM, or Google in our daily lives and indeed in our code/spaces,¹⁰⁷ heterogeneous software engineering can illuminate avenues of resistance. It is true that in the case of Vista, especially in the limited sense we offer here, the major disruption came from end-users, and so we might conclude by noting that consumer advocacy and "consumer rights" will be the most effective way to resist the hegemony of these corporations.¹⁰⁸ However, end-users may not have had this opportunity if there wasn't a rupture in Microsoft's network due to Intel's 915 chipset. Moreover, consumer sovereignty is not the only way to resist software oligopolies. Our analysis reveals that public regulation of these companies, particularly in terms of their near-collusive relationships with oligopolistic hardware manufacturers (in this case, as between Microsoft and Intel), may be effective. Of course, free and open source software (FOSS) challenges this hegemony, but our analysis reminds makers of FOSS that such software must associate end-users, who have been enculturated in a world of graphical

abstractions and an ideology of user-friendliness; end-users don't often like to hear "RTFM." Or, more radically, if FOSS software alternatives are somewhat effective, perhaps coupling them with free and open source hardware production and distribution – maybe starting in some universities or in worker-owned cooperatives – would help dissociate oligopolies. All the while, of course, these objects must be designed for use and for users, computer literate or otherwise. For anyone exploring alternatives, the heterogeneous software engineering method may be useful in theorizing or building the complex, contingent networks that must be navigated in order to meet the challenges laid down by the fundamental works of software studies.

Bibliography

- Bird, C., B. Murphy, N. Nagappan, and T. Zimmermann. "Empirical Software Engineering at Microsoft Research." In *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*, 143–150, 2011.
- Boehm, B. "A View of 20th and 21st Century Software Engineering." In *Proceedings of the 28th International Conference on Software Engineering*, 12–29, 2006.
- Brooks, Frederick P. *The Mythical Man-Month: Essays on Software Engineering*, Anniversary Edition. 2nd ed. Addison-Wesley Professional, 1995.
- Callon, M., and B. Latour. "Unscrewing the Big Leviathan: How Actors Macrostructure Reality and How Sociologists Help Them to Do So." In *Advances in Social Theory and Methodology Toward an Integration of Micro and Macro Sociologies*, 277–303, 1981.
- Campbell-Kelly, Martin. *From Airline Reservations to Sonic the Hedgehog: a History of the Software Industry*. 1st MIT Press pbk. ed. Cambridge Mass.: MIT Press, 2004.
- Crutzen, Cecile, and Erna Kotkamp. "Object Orientation." In *Software studies: a lexicon*, edited by Matthew Fuller, 200–207. Cambridge, Mass.: MIT Press, 2008.
- Cusumano, Michael A. *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets and Manages People*. 1st Touchstone Ed. Free Press, 1998.
- . "What Road Ahead for Microsoft the Company?" *Communications of the ACM* 50, no. 2 (February 1, 2007): 15.
- Davis, Michael. "Will Software Engineering Ever Be Engineering?" *Communications of the ACM* 54, no. 11 (November 1, 2011): 32–34.
- DeJean, David. "Are You Ready For Vista Graphics?" Magazine. *Information Week*, November 1, 2006. http://www.informationweek.com/news/193500969?printer_friendly=this-page.
- DeLuca, K. "Articulation Theory: A Discursive Grounding for Rhetorical Practice." *Philosophy & Rhetoric* 32, no. 4 (1999): 334–348.
- Fuller, Matthew. *Behind the blip : essays on the culture of software*. Brooklyn, NY, USA: Autonomedia, 2003.
- . *Software studies: a lexicon*. Cambridge, Mass.: MIT Press, 2008.
- Gehl, Robert W. "Real (software) Abstractions: On the Rise of Facebook and the Fall of Myspace." *Social Text* 30, no. 2 111 (2012): 99–119.
- Goffey, Andrew. "Algorithm." In *Software studies: a lexicon*, edited by Matthew Fuller, 15–20. Cambridge, Mass.: MIT Press, 2008.
- "Graphics — Windows Vista Support FAQ." Intel, November 2, 2006. <http://www.intel.com/support/graphics/sb/CS-023606.htm>.
- Greene, Ronald W. "Rhetorical Materialism: The Rhetorical Subject and the General Intellect." In *Rhetoric, Materiality, and Politics*, edited by Barbara Biesecker and John Louis Lucaites, 43–65. New York: Peter Lang, 2009.
- Gruman, Galen. "The Campaign to Save Windows XP." Magazine. *InfoWorld*, October 3, 2008. <http://www.infoworld.com/save-xp>.
- Guth, Robert. "Battling Google, Microsoft Changes How It Builds Software." *Wall Street Journal*, September 23, 2005, sec. Leader (u.s.). <http://online.wsj.com/article/0,,SB112743680328349448,00.html>.
- Hall, Stuart. "Cultural Studies: Two Paradigms." *Media, Culture, and Society* 2, no. 1 (1980).
- Hayes, B. "Computing Science: Reverse Engineering." *American Scientist* 94, no. 2 (2006): 107–111.
- Hetherington, Keith. "From Blindness to Blindness: Museums, Heterogeneity and the Subject." In *Actor Network Theory and After*, edited by John Law and John Hassard, 51 – 73. Oxford [England] ; Malden MA: Blackwell/Sociological Review, 1999.
- Hruska, Joel. "\$1.5 Billion Microsoft Vista-Capable Booty Hardly Ill-gotten." Blog. *Ars*

- Technica*, 2009. <http://arstechnica.com/hardware/news/2009/01/1-5-billion-microsoft-vista-capable-booty-hardly-ill-gotten.ars>.
- . "The Vista Capable Mess: Intel Pushes, Microsoft Bends." Blog. *Ars Technica*, 2009. <http://arstechnica.com/hardware/news/2008/03/the-vista-capable-debacle-intel-pushes-microsoft-bends.ars>
- Kelley V. Microsoft Corporation, FRD 544 US Dist. 1 (2009).
- Kelley V. Microsoft Corporation, FRD 544 US Dist. 1 (2009).
- Kirschenbaum, Matthew G. *Mechanisms: New Media and the Forensic Imagination*. The MIT Press, 2012.
- Kirschenbaum, Matthew. "Virtuality and VRML: Software Studies After Manovich", 2003. <http://www.electronicbookreview.com/thread/technocapitalism/morememory>.
- Kitchin, Rob, and Martin Dodge. *Code/space: Software and Everyday Life*. Cambridge, MA: MIT Press, 2011.
- Krige, J. "The 1984 Nobel Physics Prize for Heterogeneous Engineering." *Minerva* 39, no. 4 (2001): 425–443.
- Laclau, Ernesto, and Chantal Mouffe. *Hegemony and Socialist Strategy: Towards a Radical Democratic Politics*. London: Verso, 2001.
- Lai, Eric. "Microsoft: Vista's Momentum Will 'Accrue' for Windows 7." *InfoWorld*, May 29, 2009. <http://www.infoworld.com/d/windows/microsoft-vistas-momentum-will-accrue-windows-7-249>.
- Lash, Scott M. *Critique of Information*. 1st ed. Sage Publications Ltd, 2002.
- Latour, Bruno. *Reassembling the Social: an Introduction to Actor-network-theory*. Oxford ;New York: Oxford University Press, 2005.
- Law, John. "Notes on the Theory of the Actor-network: Ordering, Strategy, and Heterogeneity." *Systemic Practice and Action Research* 5, no. 4 (August 1992): 379–393.
- . "Technology and Heterogeneous Engineering: The Case of Portuguese Expansion." In *The Social Construction of Technological Systems: New Directions in the Sociology and History of Technology*, edited by Wiebe Bijker, Thomas Parke Hughes, and Trevor Pinch, 111–134. Cambridge Mass.: MIT Press, 1989.
- Mackenzie, Adrian. *Cutting Code: Software and Sociality*. New York: Peter Lang, 2006.
- Mahoney, Michael Sean. *Histories of Computing*. Edited by Thomas Haigh. Harvard University Press, 2011.
- Mahoney, M.S. "Finding a History for Software Engineering." *Annals of the History of Computing*, IEEE 26, no. 1 (2004): 8–19.
- Manovich, Lev. *Software Takes Command*, 2008.
- Microsoft Windows Vista Launch Human Billboard, 2007. http://www.youtube.com/watch?v=B7yRQUzhM2A&feature=youtube_gdata_player.
- Montfort, Nick, and Ian Bogost. *Racing the Beam: The Atari Video Computer System*. The MIT Press, 2009.
- Moody, Fred. *I Sing the Body Electronic: a Year with Microsoft on the Multimedia Frontier*. New York: Penguin Books, 1995.
- Naur, P., and B. Randell. "Software Engineering: Report of a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October, 1968", 1969.
- Schaefer, Peter D., and Meenakshi Gigi Durham. "On the Social Implications of Invisibility: The iMac G5 and the Effacement of the Technological Object." *Critical Studies in Media Communication* 24, no. 1 (March 2007): 39–56.
- Star, Susan Leigh. "Power, Technology and the Phenomenology of Conventions: On Being Allergic to Onions." In *A Sociology of Monsters: Essays on Power, Technology and Domination*, edited by John Law, 26 – 56. 1. publ. London u.a.: Routledge, 1991.
- Thurrott, Paul. "The Road to Windows Longhorn 2003." Blog. Paul Thurrott's Supersite for Windows, August 19, 2003. <http://www.winsupersite.com/article/product-review/the-road-to-windows-longhorn-2003>.
- . "Windows Vista Review, Part 4: The Vista Experience." Blog. Paul Thurrott's Supersite for Windows, October 6, 2010. <http://www.winsupersite.com/article/product-review/windows-vista-review-part-4-the-vista-experience>.
- Tomayko, J. E. "Software as Engineering." In *Proceedings of the International Conference on History of Computing: Software Issues*, 65–76, 2000.
- Turkle, Sherry. *Alone Together: Why We Expect More from Technology and Less from Each Other*. New York: Basic Books, 2011.
- . *Life on the Screen: Identity in the Age of the Internet*. New York: Simon & Schuster, 1995.
- Wardrip-Fruin, Noah. *Expressive Processing: Digital Fictions, Computer Games, and*

Software Studies. The MIT Press, 2012.

Whitney, Lance. "XP Still Top OS, but Windows 7 Hot on Its Trail." *CNET*, January 3, 2012. http://news.cnet.com/8301-10805_3-57351192-75/xp-still-top-os-but-windows-7-hot-on-its-trail/.

Windows Vista: Aero Flip 3D! [HD], 2009. http://www.youtube.com/watch?v=luQC4peyOv4&feature=youtube_gdata_player/.

"Windows Vista Rules for Enabling Windows Aero With Guidelines for Troubleshooting", April 13, 2009. <http://msdn.microsoft.com/en-us/windows/hardware/gg487313.aspx>.

Woolgar, Steve. "Configuring the User: The Case of Usability Trials." In *A Sociology of Monsters: Essays on Power, Technology and Domination*, edited by John Law, 57 – 99. London u.a.: Routledge, 1991.

Zachary, G. *Show-stopper! : the Breakneck Race to Create Windows NT and the Next Generation at Microsoft*. New York: Free Press, 1994.

BIOS:

Robert W. Gehl is an assistant professor in the Department of Communication at the University of Utah. He has published critiques of the architectures, aesthetics, and political economics of YouTube, MySpace, Facebook, Twitter, and blogs in *New Media and Society*, *First Monday*, *The International Journal of Cultural Studies*, *Television and New Media*, *Lateral*, and *Social Text*. His first book, *Reverse Engineering Social Media*, will appear in 2013 from Temple University Press.

Sarah A. Bell is a doctoral student of Communication at the University of Utah. Her research interests are at the intersection of information use, software design, and human values.

License:

Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0)

<https://creativecommons.org/licenses/by-nc-sa/3.0/>

References

1. Wardrip-Fruin, *Expressive Processing*, ix. (up)
2. Wardrip-Fruin, *Expressive Processing*. (up)
3. Kirschenbaum, *Mechanisms*. (up)
4. Manovich, *Software Takes Command*. (up)
5. Kirschenbaum, *Mechanisms*, ix. (up)
6. Fuller, *Behind the Blip*, 11. (up)
7. Kirschenbaum, *Mechanisms*, xii. (up)
8. Turkle, *Alone Together*, 312, note 17. (up)
9. Kirschenbaum, "Virtuality and VRML: Software Studies After Manovich." (up)
10. *Ibid.* (up)
11. *Ibid.* (up)
12. *Ibid.*; See also Kitchin and Dodge, *Code/Space: Software and Everyday Life*, 24. (up)
13. Kirschenbaum, "Virtuality and VRML: Software Studies After Manovich." (up)
14. Mackenzie, *Cutting Code*. (up)
15. Law, "Notes on the Theory of the Actor-network," 380. (up)
16. Callon and Latour, "Unscrewing the Big Leviathan." (up)
17. Law, "Notes on the Theory of the Actor-network," 380. (up)
18. Law, "Technology and Heterogeneous Engineering: The Case of Portuguese Expansion," 111. (up)
19. Krige, "The 1984 Nobel Physics Prize for Heterogeneous Engineering," 426. (up)
20. Law, "Technology and Heterogeneous Engineering: The Case of Portuguese Expansion," 112. (up)
21. As Law does in Law, "Technology and Heterogeneous Engineering: The Case of Portuguese Expansion." (up)
22. *Ibid.*, 114. (up)
23. Latour, *Reassembling the Social*, 8, emphasis in the original. (up)
24. Mackenzie, *Cutting Code*, 18. (up)
25. Law, "Notes on the Theory of the Actor-network," 384. (up)
26. Krige, "The 1984 Nobel Physics Prize for Heterogeneous Engineering," 426. (up)
27. Law, "Notes on the Theory of the Actor-network." (up)
28. Star, "Power, Technology and the Phenomenology of Conventions: On Being Allergic to Onions," 29. (up)
29. Hetherington, "From Blindness to Blindness: Museums, Heterogeneity and the Subject." (up)
30. Latour, *Reassembling the Social*, 11. (up)
31. Mahoney, *Histories of Computing*, 88. (up)

32. Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog*, 94. (up)
33. Naur and Randell, "Software Engineering" Please note that we are following the pagination used in the original report, not that used in the PDF available online. (up)
34. *Ibid.*, 13. (up)
35. See Davis, "Will Software Engineering Ever Be Engineering?," 32–34. (up)
36. Qtd. in Naur and Randell, "Software Engineering," 17. (up)
37. Qtd. in *ibid.*, 24. (up)
38. Qtd. in *ibid.*, 38. (up)
39. Qtd. in *ibid.*, 138. (up)
40. Indeed, refer to the charts on pages 21, 22, and 26-30 of Naur and Randell, "Software Engineering" to see early stages of dividing up the labor of software production and managing workers' time and costs. (up)
41. Qtd. in *ibid.*, 17. (up)
42. For example, see Brooks, *The Mythical Man-Month*; Mahoney, "Finding a History for Software Engineering." (up)
43. Qtd. in Naur and Randell, "Software Engineering," 138. (up)
44. See Opler, qtd. in *ibid.*, 124. (up)
45. Paul, qtd. in *ibid.*, 40. (up)
46. *Ibid.*, 41. (up)
47. Woolgar, "Configuring the User: The Case of Usability Trials," 73. (up)
48. Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog*; Boehm, "A View of 20th and 21st Century Software Engineering"; Mahoney, "Finding a History for Software Engineering"; Tomayko, "Software as Engineering." (up)
49. Boehm, "A View of 20th and 21st Century Software Engineering," 16. (up)
50. See Microsoft's Careers site: <http://careers.microsoft.com/careers/en/us/professions.aspx> (up)
51. *Ibid.* (up)
52. Zachary, *Show-stopper!: the Breakneck Race to Create Windows NT and the Next Generation at Microsoft*; Cusumano, *Microsoft Secrets*; Moody, *I Sing the Body Electronic*. (up)
53. See <http://research.microsoft.com/en-us/groups/ri/research/default.aspx> (up)
54. Bird et al., "Empirical Software Engineering at Microsoft Research," 2. (up)
55. Cusumano, "What Road Ahead for Microsoft the Company?"; Guth, "Battling Google, Microsoft Changes How It Builds Software." (up)
56. Naur and Randell, "Software Engineering," 35. (up)
57. Crutzen and Kotkamp, "Object Orientation," 203. (up)
58. See Hayes, "Computing Science" for a discussion of electrons and temperature in computer science. These elements affect processor speed, among other things, thus impacting the performance of software. (up)
59. See Moody, *I Sing the Body Electronic* for an account of a year with Microsoft software developers that reveals the high personal toll exacted on employees. (up)
60. Kitchin and Dodge, *Code/Space: Software and Everyday Life*, 38. (up)
61. Gehl, "Real (software) Abstractions: On the Rise of Facebook and the Fall of Myspace." (up)
62. Kelley V. Microsoft Corporation. (up)
63. See docket 93, Kelley V. Microsoft Corporation, FRD 544:2. (up)
64. See docket 131, Kelley V. Microsoft Corporation, FRD 544:82. (up)
65. For one example, go to http://www.youtube.com/watch?v=luQC4pey0v4&feature=youtu_gdata_player , Windows Vista. Or, search for "Windows Aero" in YouTube for other demos. (up)
66. See <http://www.youtube.com/watch?v=B7yRQUzhM2A>, Microsoft Windows Vista Launch Human Billboard. (up)
67. Thurrott, "Windows Vista Review, Part 4: The Vista Experience." (up)
68. Thurrott, "The Road to Windows Longhorn 2003." (up)
69. See docket 131, Kelley V. Microsoft Corporation, FRD 544:10. (up)
70. "Windows Vista Rules for Enabling Windows Aero With Guidelines for Troubleshooting." (up)
71. "Graphics — Windows Vista Support FAQ." (up)
72. Hruska, "The Vista Capable Mess: Intel Pushes, Microsoft Bends." (up)
73. That is, unless the 915 was coupled with a dedicated graphics processor. However, this was not a common occurrence, especially in laptop computers. See <http://www.intel.com/support/graphics/sb/CS-023606.htm> (up)
74. DeJean, "Are You Ready For Vista Graphics?." (up)
75. Hruska, "\$1.5 Billion Microsoft Vista-Capable Booty Hardly Ill-gotten." (up)
76. Gehl, "Real (software) Abstractions: On the Rise of Facebook and the Fall of Myspace." (up)
77. Turkle, *Life on the Screen: Identity in the Age of the Internet*, chap. A tale of two aesthetics. (up)

78. Schaefer and Durham, "On the Social Implications of Invisibility." (up)
79. See docket 131, Kelley V. Microsoft Corporation, FRD 544:7. (up)
80. Docket 131, *ibid.*, FRD 544:35. (up)
81. *Ibid.*, FRD 544:4. (up)
82. *Ibid.*, FRD 544:5. (up)
83. *Ibid.*, FRD 544:137. (up)
84. *Ibid.*, FRD 544:103. (up)
85. *Ibid.*, FRD 544:101. (up)
86. *Ibid.*, FRD 544:104. (up)
87. *Ibid.*, FRD 544:103. (up)
88. Fuller, *Software Studies: a lexicon*, 4. (up)
89. Mackenzie, *Cutting Code*, 16. (up)
90. Law, "Notes on the Theory of the Actor-network," 383. (up)
91. Lash, *Critique of Information*, 205. (up)
92. Hall, "Cultural Studies: Two Paradigms"; Laclau and Mouffe, *Hegemony and Socialist Strategy*; DeLuca, "Articulation Theory." (up)
93. Mackenzie, *Cutting Code*. (up)
94. Greene, "Rhetorical Materialism: The Rhetorical Subject and the General Intellect," 45. (up)
95. *Ibid.*, 55. (up)
96. Kelley V. Microsoft Corporation, FRD 544:5. (up)
97. *Ibid.*, FRD 544:153. (up)
98. *Ibid.*, FRD 544:154. (up)
99. Whitney, "XP Still Top OS, but Windows 7 Hot on Its Trail." (up)
100. Lai, "Microsoft." (up)
101. Star, "Power, Technology and the Phenomenology of Conventions: On Being Allergic to Onions." (up)
102. Of course, this wasn't the only court battle over Vista. The European Union's anti-trust inquiries into Microsoft and its later monitoring of Vista for compliance likely was a factor in dissociating Vista. Because we have focused on Aero, we have largely bracketed off events such as this. (up)
103. Law, "Notes on the Theory of the Actor-network," 385. (up)
104. Specifically, see the concluding chapter of Montfort and Bogost, *Racing the Beam*. (up)
105. Goffey, "Algorithm," 15–16. (up)
106. See note 1, and Fuller, *Behind the Blip*, 11. (up)
107. Kitchin and Dodge, *Code/Space: Software and Everyday Life*. (up)
108. Indeed, very often altering consumption patterns (usually by the panacea of "raising awareness") is framed as resistance to capitalist excesses. In the case of Vista, especially in reaction to its use of Digital Rights Management, one such movement was the "Bad Vista" protests. See the Free Software Foundation's badvista.org. (up)

Series Navigation

<< Editorial Issue TwoText, Speech, Machine: Metaphors for Computer Code in the Law

>>

Tags: Issue two