Organization Science

Vol. 21, No. 3, May–June 2010, pp. 713–730
issn 1047-7039 | eissn 1526-5455 | 10 | 2103 | 0713

informs®

doi 10.1287/orsc.1090.0473
© 2010 INFORMS

Minding the Gaps: Understanding Technology Interdependence and Coordination in Knowledge Work

Diane E. Bailey
School of Information, University of Texas at Austin, Austin, Texas 78701,
diane.bailey@ischool.utexas.edu

Paul M. Leonardi
Department of Communication Studies, Department of Industrial Engineering and Management Sciences,
Northwestern University, Evanston, Illinois 60208, leonardi@northwestern.edu

Jan Chong
Center for Work, Technology and Organization, Department of Management Science and Engineering,
Stanford University, Stanford, California 94305, jchong@cs.stanford.edu

In this paper, we broaden the concept of interdependence beyond its focus on task to include technology, defining *technology interdependence* as technologies' interaction with and dependence on one another in the course of carrying out work. With technologies increasingly aiding knowledge work, understanding technology interdependence may be as important as understanding task interdependence for theories of organizing, but the literature has yet to develop ways of thinking about technology interdependence or its impact on the social dynamics of work. We define a *technology gap* as the space in a workflow between two technologies wherein the output of the first technology is meant to be the input to the second one. Using data from an inductive study of two engineering occupations (hardware engineering and structural engineering), we analyzed engineers' *gap encounters* (episodes in which a technology gap appeared in the course of action) and found striking differences in how engineers minded the gaps. Hardware engineers minded the gaps by coordinating technologies via "bridges" that automated data transfers between technologies. Structural engineers, in contrast, allowed technology gaps to persist even though traversing gaps consumed significant time and effort. Our findings highlight a difference between task and technology in the degree of coordination necessary for success. Managers in our study designed policies around technology interdependence and coordination not to manage technology most efficiently, but to manage work and workers in a manner consistent with occupational structures and industry constraints. We discuss the implications of our findings for theories of organizing work.

Key words: interdependence; technology; coordination; knowledge work; engineering
History: Published online in *Articles in Advance* September 25, 2009.

Introduction

Interdependence has long been central to theories of how to organize work. Researchers have typically conceptualized interdependence as the extent to which an organization's tasks require its members to work with one another (Mohr 1971, Thompson 1967) and have focused on people's actions in relation to others (Bachrach et al. 2006, Shea and Guzzo 1987, Van der Vegt and Janssen 2003). Along these lines, Guzzo and Shea (1992, p. 296) defined task interdependence as the extent to which "group members must interact and depend on each other in order for the group to accomplish its work." A primary mechanism for managing task interdependence is coordination (Rico et al. 2008, Wageman 1995), explicitly via planning and communication (e.g., Faraj and Sproull 2000) or implicitly via anticipation of others' needs (e.g., Espinosa et al. 2004). These studies indicate that both planned and emergent structures coordinate the efforts of people who work interdependently on tasks and that effective coordination efforts often lead to important performance outcomes in organizations. Beyond performance, research has highlighted the effect of task interdependence on behaviors and attitudes that arise when coordinating work such as cooperation, learning, citizenship, helping, motivation, and satisfaction (Spriggs et al. 2000, Van der Vegt et al. 2003, Wageman 1995, Wageman and Baker 1997).

The focus in the organizations literature on coordination among *people* as they work interdependently has left considerations of interdependence among *technologies* engaged in the task largely by the wayside. To find discussions of interdependence among technologies, one must turn to research on production and operations management. This literature has recognized that, with the introduction of automated processes and computerized technologies such as numerically controlled machine tools and robotic welders, many tasks in manufacturing systems are today performed by machine. Consequently, interdependence among people has come to be complemented by, if not wholly replaced by, interdependence among technologies. This interdependence among technologies affects production outcomes so strongly that assuring that the output from one technology can be effortlessly used as input for the next one is now essential in countless manufacturing settings (Gray et al. 1993). Production researchers pay considerable attention to problems posed by technology interdependence (e.g., Carrillo and Gaimon 2000, Naveh and Erez 2004, Zantek et al. 2002) and advocate solutions that call for more coordination among technologies, such as multifunctional equipment and machines that can accept diverse inputs (Ecker and Gupta 2005, Sinreich and Nelkenbaum 2006). Overall, this research suggests that efficiency gains in manufacturing are increasingly achieved by successfully managing interdependence among technologies.

Technology's growing role in accomplishing work is by no means limited to production environments. In today's "knowledge economy," computer and information technologies are transforming service and white-collar "knowledge work." For example, travelers can now book reservations online in the absence of a travel agent, accessing through one portal the sites of airlines, hotels, and car rental companies (Campbell-Kelly 2003). Similarly, automatic teller machines have taken on a broad set of tasks that were once performed exclusively by bank tellers (Morisi 1996). Overall, computer and information systems have automated many tasks in clerical work (Zuboff 1988), and computer computation, simulation, and analysis tools increasingly aid engineers, doctors, architects, financial analysts, and other professionals in their everyday work (Boland et al. 2007, Staum 2001, Streufert et al. 2001).

As more and more technologies are introduced into service and knowledge work, interdependence among them is likely to spring up: The closer one technology lies to another along the path of work actions, the more likely interdependence will arise in the form of interface formats, information requirements, and the like. We refer to such interdependence as *technology interdependence*. In a manner parallel to task interdependence, we define technology interdependence as technologies' interaction with and dependence on one another in the course of carrying out work. As workplace technologies become more prevalent in service and knowledge work, understanding technology interdependence may be as important for theories of organizing as understanding task interdependencies. Not only may technology interdependence strongly affect performance outcomes in service and knowledge work just as it has in production work, it may also alter existing task interdependence—and associated behavioral and attitudinal outcomes—as workers' roles and tasks change. However, organizational scholars have yet to develop ways of thinking about technology interdependence or its impact on the social dynamics of work.

To begin this development, this paper inductively explores technology interdependence in knowledge work. In the course of this exploration, we address three questions based on our grounded study of technology use among engineers across two occupations and four firms. Given the lack of prior research in this area, the first question asks, *what does technology interdependence "look like" in knowledge work*? In essence, it asks how scholars can conceptualize, measure, and analyze technology interdependence. Researchers typically measure task interdependence by asking workers to rate on Likert-type scales statements such as "I have to work closely with my team members to do my work properly" (Van der Vegt and Van de Vliert 2005, p. 78). Because technologies cannot answer such questions without tracking mechanisms, we require a different approach. We begin by tracing how technologies that the engineers use feed into one another. Specifically, we document whether and how the output of one technology was used as the input for the next. We develop the concept of a *technology gap* to describe the "space" between two interdependent technologies: a technology gap signals a transfer of work product from one technology on which an operation has been completed to another on which the next operation is to be carried out.

We identify in our field notes 310 *gap encounters*, or episodes in which an engineer, in the course of his work, came to the edge of a technology gap that he had to traverse. By examining the direction of workflow across the gap (forward or backward), we find that technologies in both occupations exhibited, to use Thompson's (1967) terms, considerable *sequential* and some *reciprocal* interdependence. We categorize gaps by their "width," a measure of how difficult traversal of the gap was for the engineer, to reveal differences by occupation in the distribution of wide and narrow gaps in the forward and backward directions. By examining how many substitutable technologies were available to the engineers for the completion of any task, we can speak to the prevalence of *pooled* technology interdependence as well. We also show that interdependence among technologies in both settings was largely distinct from task interdependence among people. We find that although both occupations exhibited what we would call high technology interdependence, differences in how that interdependence was manifested across occupations suggest that the experience of technology interdependence might vary considerably.

We explore this possibility in our second question: *How do knowledge workers experience and deal with technology interdependence*? In particular, we focus on how engineers minded technology gaps, ultimately developing a typology of gap-traversal strategies used by the engineers we studied. These strategies included *navigating*, *bridging*, *crossing*, *expanding*, *bypassing*, and *standing still* at gaps. We find that one occupation actively and purposefully narrowed gaps by creating "bridges" that linked technologies. Bridges largely automated the transfer of data from one technology to the next and, by their permanence, made future traversals easier and speedier for all engineers in the firm. In this manner, bridges alleviated many of the problems associated with technology interdependence. The other occupation, in contrast, minded gaps by intentionally creating few bridges across them. The absence of bridges forced engineers in this occupation to arduously navigate gaps that remained wide for one and all. In this occupation, the problems associated with technology interdependence in the everyday carrying out of tasks rarely waned.

In sum, work in the first occupation reflected a high level of coordination among technologies, whereas work in the second occupation reflected a low level of coordination among technologies. This finding is striking because coordination has always been tightly tied to the concept of task interdependence. Some scholars measure task interdependence by asking questions about coordination (e.g., Langfred 2007, Linden et al. 1997, Sharma and Yetton 2003); other scholars contend that coordination is required for successful task performance when task interdependence is high (de Jong et al. 2007, Rico et al. 2008, Wageman 1995). This tight coupling between coordination and task interdependence stands in contrast to our findings: Among the engineers we studied, high technology interdependence was associated with high technology coordination in only one occupation; engineers in the other occupation also faced considerable technology interdependence, but intentionally eschewed coordinating their workplace technologies in the course of successfully performing their work.

This difference between technology interdependence and task interdependence with respect to the degree of coordination prompts our third question: *What factors shape the development of low versus high coordination practices and mechanisms for handling technology interdependence in knowledge work*? In other words, if high coordination is tied to high *task* interdependence, why is high coordination not similarly tied to high *technology* interdependence? We find that the decision to tightly or loosely couple coordination and technology interdependence was influenced by a mix of work characteristics, occupational structures, and industry constraints. In hardware engineering, factors such as high technology costs and the ability to entrust logic testing of models to technology rendered streamlining data transfer among technologies a successful strategy; in this case, high technology interdependence was tightly coupled with coordination. In structural engineering, in contrast, liability concerns, difficulties in assessing the soundness of models, and the desire to develop design knowledge among engineers worked against such streamlining, resulting in little coordination among interdependent technologies.

Our findings are an important contribution to the organization studies literature because they suggest that modern tasks may feature interdependence among technologies in addition to, and distinct from, interdependence among people. Moreover, the requirements for effectiveness may be different for each type of interdependence: High levels of task interdependence may call for high coordination, but high levels of technology interdependence may not necessarily do so. In our study, managers designed policies around technology interdependence not to manage technology most efficiently, but to manage work and workers in a manner consistent with their occupational environment.

Methods

Research Design

We investigated technology interdependence in knowledge work in two occupations: structural engineers in building design and hardware engineers in computer chip design. Structural engineers specify the materials, shapes, and sizes of the beams, columns, and other elements that transfer loads to the ground to prevent buildings from collapsing. Hardware engineers craft the logic of microprocessor cores, buses, and other chip components for products they bring to market. These engineers write code in high-level programming languages that specifies how components will handle instructions; they also write programs to verify proper component functioning.

Our data are drawn from two structural design firms and two chip design firms in the San Francisco Bay area. Across the four firms, we shadowed 12 structural engineers and 15 hardware engineers on repeated occasions. To maintain a focus on engineering rather than management tasks, we targeted junior and mid-level engineers. Between 1999 and 2001 we conducted 61 observations of structural engineers and 65 observations of hardware engineers.

In preliminary interviews with senior managers, we learned that technologies aided nearly every task in both occupations. Structural engineers created designs via computation and equation solving on pocket calculators and via modeling in computer-aided drawing (CAD) software packages. They used information from design manuals, building codes, textbooks, and past project records as they analyzed models in Excel or computational software programs. Hardware engineers wrote code and used checking programs to verify logic and syntax. Verified code was entered with other component code and specification files into simulators for testing. Simulation results could be read in text form or fed into wave-form generators for interpretation. The prevalence of technology use in the course of work tasks in both domains indicated the likelihood of considerable technology interdependence. Other aspects of the technology infrastructures, however, such as differences in the cost of technologies, suggested that technology interdependence might differ substantially across the two occupations, thus providing a useful comparison.

Data Collection

Engineering work has been notoriously difficult to study (Barley 2005, Downey et al. 1989). Engineers carry out tasks with few everyday correlates and use sophisticated technologies whose functioning is not easily gleaned. An engineer may simultaneously use multiple technologies, engage in discussion, and consult a variety of artifacts. To meet these challenges, we interviewed a manager or engineer in each firm so that he could outline the major phases of design and the tasks undertaken in them. From these descriptions, we built timelines that specified the normal progression of design. We studied textbooks and design manuals that explained the analyses required for design in each discipline; doing so helped us to understand the modeling and computation steps that engineers would perform. We asked engineers (not in our sample) to tutor us in greater detail on a set of primary work tasks and technologies so that we would be better prepared to link textbook theory to actual practice in our observations.

We wrote detailed field notes during our observations, in which we recorded the engineer's every action, including his typed commands, computer program starts, switches and stops, and use of artifacts. We asked for screenshots when engineers worked on the computer, made photocopies of physical documents they used (e.g., book pages), and sketched other artifacts that could not be captured by computer or copier (e.g., drafting tools). We audiotaped conversations whenever engineers spoke too quickly or too technically for us to keep pace by hand. We later entered transcriptions of the tapes into the text of the notes at the point they occurred. Collecting so many types of data enabled us to prepare field notes that describe actions, conversations, and visual images simultaneously and thus produce a record not only of what engineers did and said, but also of what they worked on and created. We wrote appendices for each day's field notes that describe the artifacts we collected, including how and why the engineer used the artifact. We began expanding our field notes immediately after each observation; completing a full narrative of each three to four-hour observation session took between two and two and a half days.

The intensity of our techniques for collecting and recording data prompted us to study the four firms sequentially so that the research could remain manageable. Thus, we began and completed the research on structural engineering before moving to hardware engineering. Three graduate students assisted with the fieldwork in structural engineering and two others assisted with the research on hardware engineering. The first author did fieldwork in all four sites and completed 50% of the observations in both occupations to ensure that we would retain a deep understanding of each site when the students graduated. Consistency and coordination were especially important in a project spanning multiple years and sites. The first author trained all team members who did fieldwork and reviewed every set of notes that they produced for thoroughness, technical accuracy, and conformity to formats.

Data Analysis

The first task of data analysis was to conceptualize technology interdependence in a manner that would allow us to identify and isolate incidents in the field notes in which interdependence was evident. We adopted the recommendation of Glaser (1978) to develop analytic constructs, grounded in our rich observational data, that would aid us in systematic analysis. Following Glaser's practice of "theoretical sensitivity," and recognizing from our fieldwork that technology interdependence is often manifested at the intersection of technologies, we first developed the concept of *technology gap*. We defined a *technology gap* as the space in a workflow between one technology and a second technology wherein the output of the first technology was meant to be the input to the second one. An example of a technology gap existed in structural engineering between CAD software (for drawing solutions) and analysis software (for testing solutions). Transferring the CAD model from the CAD software to the analysis software required human effort; no single command ran first one application and then the other.

In identifying technology gaps, we counted as technology any artifact that an engineer used in the course of his work that provided output or allowed input. Thus, books and manuals were included because their content often was the output that engineers entered into other technologies; most software programs were also included. Work implements not counted included pencils, erasers, straight edges, and other implements that had less discernible input or output. Also not counted were functions within technologies, such as a spell-checking function in a word processor.[1]

We next defined a *gap encounter* as an episode in which a technology gap appeared in the course of action. That is to say, a gap encounter occurred when an engineer sought to transfer the output of one technology so that it might serve as the input for a second one. Gap encounters often appeared when engineers switched from one technology to another and in doing so carried forward some piece of work from the first technology for further manipulation by the second one.

The important aspect of gap encounters was the transfer of output to input: the focus was on the path of work across technologies and hence technology interdependence, not between people and hence task interdependence. Thus, gap encounters could have involved multiple individuals; in our observations, however, most gap encounters featured a single individual who operated both technologies.

Detecting engineers' encounters with gaps was straightforward in observations of structural engineers, whose work was often done with paper, pencil, and a calculator. A structural engineer launching a software application, for example, entailed an obvious turn in posture toward the keyboard; similarly, using a design manual required reaching for it on a bookshelf. In hardware engineering, almost all work was completed on the computer, so most technologies were launched by command, with few distinguishable body movements to signal switches among technologies. We therefore positioned ourselves during observations so that we could see all keystrokes. In addition, we detected early cues of switches whenever an engineer began to prepare her data for output. Such preparation may have entailed such distinct steps as making selections from pull-down menus, running an external script, or processing the data format manually (e.g., by adjusting it row by row in a file). All these actions served as behavioral signals that a technology gap might soon be encountered.

We began our analysis by isolating the gap encounters in four sets of field notes, two from each occupation. We examined these encounters to determine which of their attributes might best help us conceptualize technology interdependence. We made detailed lists of such attributes, examined the lists with respect to gap encounters pulled from a broad cross section of our data, and revised the lists to reflect those attributes that were common to all gap encounters (Strauss and Corbin 1998). Ultimately, we identified three key attributes that would help us distinguish among and compare encounters: the direction of the flow of work across the gap, the size of the gap, and the strategy the engineer undertook to deal with the gap. Each attribute had more than one possible value; for example, we observed many strategies for contending with gaps. We define the attributes and their observed values in the next section.

These attributes and their values became the codes that we used to code the gap encounters in the remaining field notes. In total, we identified 127 gap encounters in 176 hours of observation in structural engineering and 183 encounters in 178 hours of observation in hardware engineering. The first author coded each of these 310 gap encounters using ATLAS.ti©. In the course of coding, we found that some engineers traversed a given gap many times, whereas other engineers traversed it only once in completing the same task. One engineer, for example, sent his code for error checking after writing each 20 lines of code, and another waited until she had completed her entire code file. To account for these differences in behavior, we coded a gap encounter only once per observation if the engineer was transferring the same object (e.g., the same model or the same piece of code) across the same gap between the same two technologies. In other words, we opted not to inflate the number of gap encounters caused by engineers' variation in work practices, but to render counts that reflected how often the task was performed.

Technology interdependence is not completely described by the three attributes of technology gaps on which we focused, but analyzing these attributes helped us conceptualize and document, in particular, sequential and reciprocal interdependence. To investigate pooled interdependence, we examined the number of substitutable technologies available to complete each task. In addition, to enhance our understanding of technology interdependence in all its forms, we drew heavily on our field notes for incidents and conversations that would help us deepen our insights, interpret our findings, and support our conclusions.

Technology Interdependence in Two Occupations

What Technology Interdependence "Looks Like"

Technology interdependence in the engineering work we observed paralleled the flow of work tasks as engineers moved forward through stages of design and analysis. In this task progression, the output of one technology became the input to a second technology as the engineer's product (be it a drawing, a computational model, or code) advanced toward completion. Each transfer of the product from one technology to the next indicated a technology gap. For example, when a hardware engineer finished writing his component code, he might submit it to a simulator for testing. The serial traversal of technology gaps in this manner spoke to the *sequential* interdependence among technologies in both occupations.

Engineering work also flowed backward. A backward flow was almost always generated by feedback. Backward flows in the form of feedback occurred, for example, when a testing program detected problems in a model or a piece of code. The engineer would use this feedback to modify the model or code in the relevant technology he had used to create his design. Feedback varied in its explicitness: Some testing programs merely detected errors, others pointed out error location, and still others diagnosed possible causes, and a rare few suggested corrections. In each case, the feedback was intended to serve as input to a technology with which the engineer would modify the model or code. Because this technology was typically the same one the engineer used to create the model or code, the presence of technology

Table 1 Direction of Workflow Across Gaps by Occupation

	Structural engineering		Hardware engineering	
	N	Percent	*N*	Percent
Forward	101	80	154	84
Backward	26	20	29	16
Total	127	100	183	100

gaps in the backward flow reflected *reciprocal* interdependence among technologies in these two occupations.

Table 1 shows that in both occupations, engineers encountered the most gaps as they progressed in a forward direction along the sequentially organized flow of engineering work. In structural engineering, 80% of all encountered gaps arose in the forward direction; in hardware engineering 84% arose in the same direction. At 20% and 16% of all gaps, respectively, backward gaps indicated a degree of reciprocal interdependence among technologies in both structural and hardware engineering.

Technology gaps in either direction required human intervention for traversal. This intervention was more strenuous for some gaps than for others. As a first step in characterizing the difficulty associated with traversal, we denoted the size of the gap, or the distance between the two technologies, by categorizing gaps as being *wide* or *narrow*. We defined wide gaps as being more difficult to traverse than narrow gaps. We assessed gap size based on a combination of the amount of time it took the engineer to traverse the gap and the number of distinct steps or actions that were required. Hence, a gap was *wide* when its traversal required a significant amount of time (as much as an entire afternoon) and possibly multiple actions (e.g., manually transferring data points one by one across several input screens). We called a gap *narrow* when the output of the first technology became the input to the second one in a very short time (as little as a few seconds and often in under a minute), with few related actions (often simply typing a single command).

A wide gap is illustrated in the example of Cindy, a structural engineer who designed her steel beams in AutoCAD software and then analyzed their deflection in a program called RAM SBeam. Because there was no easy mechanism for transferring data from AutoCAD to RAM SBeam, Cindy had to enter the values manually. The transfer was further complicated by the fact that the AutoCAD values were in different units than RAM SBeam would accept, as Cindy explained:

> I get the dimensions from CAD [she points to the text lines at the bottom of her screen, where numbers appear], but RAM SBeam is in kips^2 per feet and CAD is in pounds per square foot, so I use my calculator to do the conversion. [She picks up her pocket calculator and enters some values and then types the result into the entry field of the RAM SBeam window.]

Table 2 Size of Gap by Direction and Occupation

	Structural engineering		Hardware engineering	
	N	Percent	*N*	Percent
Size of gap				
Wide	118	93	76	42
Narrow	9	7	107	58
Total	127	100	183	100
Forward distribution				
Wide	92	91	47	31
Narrow	9	9	107	69
Total	101	100	154	100
Backward distribution				
Wide	26	100	29	100
Narrow	0	0	0	0
Total	26	100	29	100

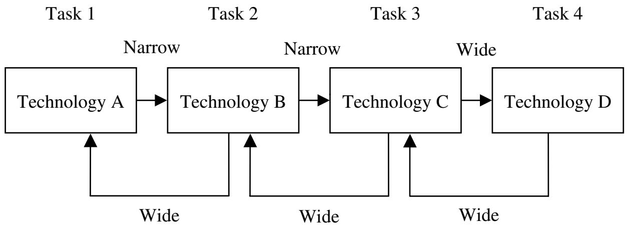
In an example of a narrow gap, Eric, a hardware engineer, created a diagnostic test and was ready to run it on a microprocessor configuration. To do so, he simply typed a command, as described in our field notes: "At the prompt in the lower left window, he types: make vcs.log DIAG-103."

Table 2 shows the distribution of wide and narrow gaps across the two occupations. Wide gaps outstripped narrow gaps in structural engineering by a large margin, constituting 93% of all gap encounters in this field. In hardware engineering, the balance was tipped in the opposite direction, with engineers encountering narrow gaps (58%) more frequently than wide gaps (42%). Table 2 further categorizes wide and narrow gaps in terms of the flow of work. In structural engineering, more than 90% of the gaps in the forward direction were wide. In contrast, in hardware engineering, fewer than a third of the forward gaps were wide. Compared with structural engineers, hardware engineers switched with ease from one technology to another in the forward progression of work. In both occupations, all gaps in the backward direction were wide.

To complete the picture of technology interdependence in each occupation, we next considered whether *pooled* interdependence existed among the technologies. This endeavor required that we examine the degree to which an array of substitutable technologies was available to aid with any given task. If different technologies were available, there would be pooled interdependence.

In hardware engineering, we found that most tasks were associated with a single technology unique to that task. Consequently, engineers rarely had a choice among technologies in their work. To verify code via simulation, for example, engineers in one firm we studied used a Verilog simulator made by a supplier named Synopsys; Cadence as well as other design and analysis technology suppliers also sold Verilog simulators, but that firm owned only the Synopsys simulator. The same was

Figure 1 Technology Interdependence in Hardware Engineering



true for the tasks that fell before and after simulation, which meant that the gap in front of and the gap behind the simulator each had a single technology on either side. Hardware engineers explained that the high cost of their work technologies—as much as a million dollars in some cases—was a strong deterrent to purchasing multiple equivalent technologies to perform any given task. Pooled technology interdependence was, therefore, largely absent in hardware engineering.

In structural engineering, we found the opposite situation: Numerous technologies existed for most tasks and engineers could choose which technologies to use. For example, to determine the size of beams, the engineers could use a commercial product called RISA-3D or an Excel spreadsheet created by a colleague that accepted input for predetermined formulas. Alternatively, the engineers could look up values for material strength in design manuals and then use their calculator to determine the beam size "from scratch" by writing down and solving a series of equations, with the strength values as input. The structural engineers noted that many technologies were available for each task because each technology cost relatively little—the most expensive technology cost $25,000—and some technologies, given to the engineers by professors back in graduate school, were nearly free. Moreover, much of the foundational knowledge that underpinned the algorithms embedded in the technologies retained its value over time (e.g., the physics was still valid, many of the material properties were unchanged), which meant that old technologies remained useful even as new technologies entered the firm. Hence, few technologies were retired. Thus, in structural engineering, pooled technology interdependence was considerable.

We pull together our findings of flow direction, gap width, and technology substitution in graphic representations of technology interdependence in each occupation. Figure 1 provides a simplified representation for hardware engineering. Four technologies (A, B, C, D), one for each of four tasks (1, 2, 3, 4), lie in progression, reflecting sequential interdependence. The forward gaps between these technologies are mostly narrow; about one-third are wide. All the backward gaps, which indicate reciprocal interdependence, are wide. Figure 2 provides a simplified representation of technology interdependence in structural engineering. Like Figure 1,

Figure 2 Technology Interdependence in Structural Engineering

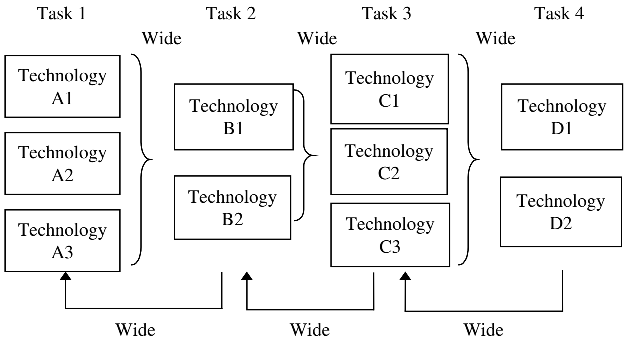


Figure 2 shows four tasks in progression, but in this case multiple technologies are available for each task: three technologies (A1, A2, A3) for the first task, two (B1, B2) for the second, three for the third (C1, C2, C3), and two for the fourth (D1, D2). These multiple technologies for each task reflect pooled interdependence that is missing in the hardware engineering representation. Forward and backward gaps are wide in structural engineering. The representations in Figures 1 and 2 reflect high levels of technology interdependence: Almost all tasks in both occupations were aided by technology, and the output of one technology became the input for another technology as the work progressed.

Task interdependence looked quite different from technology interdependence in both occupations. In hardware engineering, junior and mid-level hardware engineers worked in project groups of 5–10 engineers, managed by a senior engineer. Each engineer was assigned a particular microprocessor component; his job was to write and test the code that would model that component. Input and output specifications were made early to reduce sequential task interdependence among the engineers; engineers could begin work on their component without completed code on other components because they assumed that the output of other engineers' models would conform to their models' input requirements per these specifications. Development could not begin, however, until these specifications were determined.

Combined, the engineers' component models formed the coded representation of a given version of the microprocessor and its peripherals, which evidenced pooled task interdependence. A separate group typically tested the complete representation, but each hardware engineer routinely built the full model when testing her own component to make sure that it interacted properly with other components. In these component tests, older or less detailed versions of other component models were generally sufficient, thereby lessening sequential task interdependence by removing any need to wait for another engineer to complete his work before testing one's own. Failure in these tests sometimes prompted an engineer to suspect that the problem lay not in her component's model but in someone else's, a suspicion that could prompt feedback to the other engineer, whose subsequent modification of his model would reflect reciprocal task interdependence.

Hardware engineering thus exhibited aspects of sequential, reciprocal, and pooled task interdependence, but practices (such as early input/output specification) were developed such that pooled task interdependence primarily characterized daily work. Under this design, hardware engineers, although specialists in the design of particular types of components, carried out very similar kinds of tasks and used a common set of technologies to create and test those components.

Overall, task interdependence was distinct from technology interdependence in hardware engineering. Each engineer individually made her way through the sequential progression of technologies as she completed the full set of tasks on her own component. Her peers similarly made their way through the same progression of technologies as they completed the same set of tasks on their components. Along this progression, each engineer's work product traversed technology gaps as it moved from one technology to another. In traversing these gaps, products did not simultaneously move from one engineer to another; rather, a single engineer shepherded a product from the beginning to the end of the hardware engineering design flow. As a result, each engineer experienced the full set of technology gaps, and these gaps were identical to those experienced by the other hardware engineers in the firm. Task interdependence was primarily pooled (engineers worked separately on components that were later combined to form the whole product); technology interdependence was primarily sequential (each engineer's product moved from one technology to the next as work progressed) and experienced identically by all hardware engineers.

A similar situation existed in structural engineering. Working in groups of two to four engineers, structural engineers created drawings that illustrated their design solution and conducted analyses to test its soundness. Senior engineers supervised the work of junior and mid-level engineers, who tended to divide their work such that they each performed the same tasks, but on different parts of the building. For example, one engineer might design the bottom two floors of a building while another engineer designed the top two floors. Each engineer built models for and ran initial analyses on the floors she designed.

Although the physics of building structures compels each floor to bear the load of all the floors above it, a fact that might suggest the sequential design of a building from its uppermost floor to its lowermost, in practice engineers used estimates of other floor loads to decrease sequential task interdependence among themselves. Some sequential task interdependence did remain; for example, owners had to specify materials before floor design could begin. However, on gaining this information, structural engineers worked largely independently on their assigned floors. When the designs for all the floors were complete, one engineer pooled all the floor designs into a model of the entire building and conducted final analyses. Results of these analyses could serve as feedback to the engineers who designed, and had to modify, particular floors.

Like hardware engineering, structural engineering work simultaneously exhibited aspects of sequential, reciprocal, and pooled task interdependence, but the work was organized such that pooled task interdependence characterized the bulk of daily activity. Under this organization, engineers were generalists who used a common set of technologies in the completion of their work. As in hardware engineering, task interdependence was largely distinct from technology interdependence because each engineer made her way through the sequential progression of interdependent technologies as she completed the full set of tasks on her own floors and in other building components. The specific technology gaps the engineers encountered varied to the extent that their choices among substitutable technologies for a given task varied, but all engineers faced the same number of gaps in completing the same tasks.

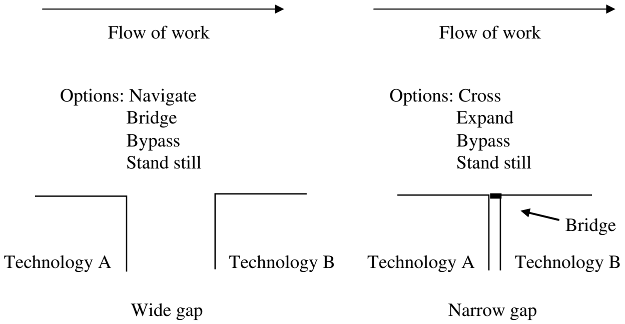
Although technology interdependence was high and distinct from task interdependence in both occupations, there were differences in terms of the degree of technology substitution and the presence of narrow gaps. These differences suggested that the manifestation of high technology interdependence varied by occupation, prompting us to explore how both hardware and structural engineers experienced and dealt with technology interdependence in the course of their everyday work.

Experiencing and Dealing with Technology Interdependence

In the course of everyday work, pooled technology interdependence posed few direct problems for structural engineers because having multiple technologies from which to choose was not terribly troublesome (most engineers were happy to select their personal favorites). In contrast, sequential and reciprocal technology interdependence presented structural and hardware engineers with various kinds of problems and dilemmas in the context of forward and backward gaps. Therefore, we examined what engineers experienced and what they did when they encountered a technology gap.

Our analysis revealed six distinct strategies that an engineer could take in dealing with a technology gap. Figure 3 displays these strategies in conjunction with the size of the gap—wide or narrow—the engineer faced. Wide gaps presented the engineer with the prospect of significant time and effort for their traversal; engineers

Figure 3 Strategies at Wide and Narrow Gaps in a Workflow



could choose from among four strategies for how to contend with them. Narrow gaps presented the prospect of an easier and speedier traversal than that of wide gaps; engineers could again choose from four strategies (two unique to narrow gaps, two common with wide gaps) in dealing with them.

In the case of a wide gap, the engineer might have decided to traverse the gap by *navigating* it. To navigate a gap was to transfer, and often transform, the output from the first technology to the input of the second technology. Navigation facilitated traversal in that one instance only. In other words, a wide gap that one individual navigated still appeared as a wide gap to the next person who encountered it.

The methods of transfer and transformation used in navigating a gap almost always required considerable human effort, typically in the form of routine actions. Engineers often referred to navigation as having to do the transfer "by hand," meaning that no simple keystroke or command call would complete the task for them. Rather, navigating required labor-intensive actions, such as cutting and pasting data, visually inspecting and comparing models, manually checking values or equations, and entering long series of separate commands at a prompt. Engineers endured, not enjoyed, navigation.

In a navigating example from structural engineering, the engineer transformed and then transferred output from AutoCAD into an analysis program called RISA-3D:

> Darren opens the text window in AutoCAD and copies some numbers from it onto a piece of scrap paper. Then he uses his calculator, slapping it first, because, he says, its batteries are low. He opens another window for RISA-3D. He explains that Risa works in feet, not inches, so he is converting distances between columns on the drawing and entering the values in the RISA data fields. He uses the calculator to make the conversion, a simple division operation.

An alternative to navigating a gap was to *bridge* it. Bridging allowed for easier traversal of the gap by lessening the effort required for transferring and transforming the output of the first technology into the input of the second one. In general, bridging meant automating some portion of the steps required to transfer and transform output to input, perhaps by writing a script to carry out a series of commands. Establishing a bridge simplified future traversals by converting a wide gap into a narrow one. Because bridging laid the groundwork for future gap encounters, it often took longer than navigating, which focused only on the immediate encounter and left nothing behind for the next traversal. Therefore, an engineer who opted to bridge a gap typically weighed the benefits of time saved in future traversals against the cost of additional time spent on the present encounter. This cost notwithstanding, engineers who engaged in bridging reported enjoying the opportunity to be creative by solving the puzzle of transfer and transformation that was unique to the technologies involved. Bridge creation was not an everyday affair and thus offered a break from normal tasks.

An example of bridging is found in the case of a hardware engineer who had written a diagnostic test to run on a set of microprocessor configurations. Although scripts existed for loading and running his test on the simulator against the configurations one at a time, he wanted to run his test against each configuration multiple times. He thus wrote a script that built and repeatedly sent each configuration to the simulator with his test code. The script constituted a bridge between the text editor (in which he created his test code) and the simulator (on which he would run it), a bridge that he could use again any time he wanted to traverse this same gap.

In our observations rarely did an engineer who encountered a technology gap opt to *bypass* it or *stand still* at its edge, but these options did exist. A gap was *bypassed* when the engineer managed to complete his task without having to deal with the gap. For example, he might have bypassed a gap by selecting an alternative second technology whose input requirements were more sympathetic to the first technology's output. Alternatively, an individual might have opted to *stand still* at the edge of a gap by halting his efforts to proceed. Because an engineer who encountered a technology gap typically had some task to complete that he could not ignore—perhaps to create a document, build a model, run an analysis, or interpret data—the option to stand still was rarely observed.

In the case of narrow gaps, the strategy options were slightly different. Navigation and bridging were no longer options, presumably because the gap was narrow and could be easily traversed simply by *crossing* it. To cross a gap was to expend very little effort in guiding the output of the first technology to the second one, where it served as the input. No manipulation of the output was required, typically because the individual who crossed a gap did so via a bridge from an earlier traversal. When Eric ran his diagnostic test on a microprocessor configuration by simply typing the "make" command with

Table 3 Strategies for Dealing with Gaps by Occupation

	Structural engineering		Hardware engineering	
	N	Percent	*N*	Percent
Navigate	117	92	72	39
Bridge	0	0	3	2
Bypass	0	0	1	1
Stand still	1	1	0	0
Cross	8	6	105	57
Expand	1	1	2	1
Total	127	100	183	100

two parameters, that constituted an example of crossing a gap. He was able to feed the results of his component modeling into his diagnostic test by using a script that had been created earlier. Crossing gaps allowed engineers to proceed quickly with the rest of their work.

Rather than crossing a narrow gap, an individual could decide to *expand* it. By expanding a narrow gap, he chose to undo, if only in that single instance, whatever action previously bridged it, perhaps because he considered that action an inappropriate or inelegant way to deal with the gap. We saw a hardware engineer expand a narrow gap one day when he decided that, rather than run an existing script that would automatically send his component code to the simulators with a suite of tests, he would complete each step of the process "by hand" (i.e., he would enter individual commands one by one) so that he could determine where a problem was occurring. Theoretically, individuals could also bypass narrow gaps or stand still at their edge; we never observed these actions at narrow gaps, presumably because traversing narrow gaps was relatively easy.

In both occupations, the engineers we observed typically navigated wide gaps and crossed narrow ones. However, because wide gaps were much more common in structural engineering than in hardware engineering, engineers in that field were much more likely to navigate a gap than to cross one. Table 3 shows that structural engineers navigated 92% of the gaps they encountered; in contrast, hardware engineers only navigated 39% of their gaps. Hardware engineers more commonly crossed gaps (57%), a high rate in comparison to structural engineers (6%).

Engineers in neither occupation bridged, bypassed, expanded, or stood still at gaps very frequently. However, hardware engineers told many stories of gaps that they had bridged in the past, often proudly showing the scripts they had written, as explained by this engineer:

> Timing Design is a program to draw waveforms. In it, you have to click all over to draw something. I didn't like that. And I don't want to draw it by hand. So I wrote a little program called Text2TimingDesigner. With my little program I can write text files and it generates waves. I have written lots of little programs to do things for me. I keep them in a directory. It's laziness. You can write it once and then you have it; it automates for you.

Structural engineers related no such stories. Indeed, Table 3 indicates that hardware engineers predominantly crossed gaps (57%), whereas structural engineers rarely had that opportunity (6%). These data show that although we did not often observe hardware engineers actively bridging gaps, there is evidence that many bridges had been built because a gap could not be crossed if it was not previously bridged. In contrast, structural engineers did not routinely cross gaps because they had not previously bridged many gaps.

Overall, traversing gaps in structural engineering was an arduous affair. Most gaps were wide and thus required time and effort to navigate via the carrying out of often routine and mundane actions. In comparison, traversing gaps in hardware engineering was easy because most gaps were narrow, making crossing the most common strategy and allowing engineers to carry on quickly with the rest of their work. Our findings indicate that the width of a gap was not fixed by the technologies on either side of it; engineers could alter this width by, for example, bridging wide gaps to make them narrow. Thus, engineers seemingly had the potential to increase coordination among technologies via smoother interfaces in the form of narrow gaps; in so doing, they could arguably better manage technology interdependence. This possibility prompted our third and final question: *Why did engineers in one occupation manage high technology interdependence in ways that led to high coordination among technologies while engineers in the other occupation did not?* Specifically, why did structural engineers persist with navigating wide gaps rather than bridging them to facilitate easier traversals like the hardware engineers did?

Coordination and Technology Interdependence

We approach the question of differences in coordination by first considering what factors enabled hardware engineers to narrow the gaps among their technologies and then turning to structural engineering to see if the same or different factors existed there. In hardware engineering, the high cost of technologies worked against the possibility of having multiple equivalent technologies to perform any given task. With only one technology on each side of a technology gap, narrowing the gaps was a straightforward, if still difficult, task because complexity of the interface problem for inputs and outputs was greatly reduced. With the output from a single technology serving as the sole input to the next one, only one interface problem had to be resolved at each technology gap.

These interface problems greatly attracted the attention of hardware engineers, who explicitly recognized, frequently discussed, and overtly managed technology interdependence, going so far as to give it a name: "the design flow." We observed hardware engineers contemplating whether their design flow could support new microprocessor configurations, arguing about whether to change the flow to satisfy customer requests and mulling over how to build new flows for new microprocessor cores. Similar to efforts in production settings, concerted efforts were made to improve the efficiency of the design flow by embedding multiple functions in single technologies and purchasing new technologies to replace technologies with problematic interfaces.

A primary criterion in new technology selection was thus the technology's "fit" with the flow, namely, whether it could be placed seamlessly between two existing technologies without wreaking havoc on inputs and outputs. To learn about new technologies, hardware engineers visited vendors' booths at the industry's annual conference; we saw them preparing for these visits by discussing with their colleagues which vendors to see and what technology features to inspect. Vendors pitched new technologies, as we observed during an in-house presentation, by explaining how easily the technologies could fit in a firm's design flow. After hearing such pitches, engineers independently evaluated new technologies' fit prior to purchase, as this hardware engineer described:

> I have been testing the flow of a CAD tool called Blast Fusion from Magma [the vendor]. So far, I have mostly verified that the flow works and everything is fine, so the other projects can go ahead and use that tool.

Following technology selection, hardware engineers narrowed gaps by bridging them, namely, by writing scripts that automated the transfer of output. To automate this transfer was, in a sense, to abdicate control by forgoing the opportunity to review output. Hardware engineers felt comfortable in relinquishing this review opportunity because they were confident that unsound design as created in one technology would be detected by error-checking features in subsequent technologies. In addition, the size and complexity of hardware designs, with literally hundreds of thousands of transistors to be laid out and perhaps tens of thousands of code files called in the course of a test, made human inspection of results very difficult. The challenge was to think of all possible forms of failure (a problem known as "coverage"), not to worry that any particular problem could not be caught by technologies once it was identified and a suitable program written to test for it. As a result, most gaps in the forward design flow were narrow, meaning that someone had bridged them to reduce human involvement in transferring output.

Beyond visiting vendors' booths at the annual industry conference and inviting vendors to give in-house presentations of new technologies, hardware engineers also entered into partnerships with vendors for the development and licensing of new technology features. Hardware engineers further interacted with external parties via online bulletin boards that were hosted on independent hardware engineering community websites. Through such electronic media, engineers across competing hardware engineering firms discussed and evaluated new technologies. Information sharing among peers across firms also occurred in trade magazines, which regularly featured interviews of engineers whose firms were among the first to purchase particular technologies. Engineers in these interviews reported how well the technology fit in their firm's design flow. The engineers we observed contributed to, read, and made use of such print and online information. For example, one engineer was interviewed by a magazine about the firm's use of a new logic checking application, another engineer made a post to a Web forum about a parsing program, and a third engineer downloaded from a vendor-supported user-group website a script for bridging a gap to a file-sharing application.

Internal bridge building was also supported and formally organized. Hardware engineers were assigned to serve as gurus for specific technologies, charged with answering colleagues' questions and keeping up to date on new features and products. As part of this duty, gurus were expected to build bridges, as described by this engineer:

> Most of these tools have ways of automating tasks and writing scripts that would help you do the common tasks. And so one of the things we have done is [have] the person who is an expert in that help set up some standard scripts or templates, and in many of your applications, you could just use that set-up as is or do minor modifications to it, and you don't necessarily need to understand exactly every step of the way. There is an automation mechanism created by this expert.

In addition, systems administrators were used to maintain the computer network, to manage licenses, to prepare technology budgets, and to help assess the fit of potential new technologies in the design flow. These examples make clear that two factors shaped the goal of high coordination among interdependent technologies: (1) the belief that bridging activities automated tasks that were of marginal importance or that involved knowledge that could be codified, and (2) occupational ideas of efficiency. Furthermore, this goal was institutionalized in occupational roles. In short, managing technology interdependence in hardware engineering was a shared task among a range of individuals.

None of these bridge-building efforts, however, involved backward gaps. Backward gaps in the hardware engineering flow were left wide because feedback from testing programs typically prompted code modifications. These modifications almost always required critical engineering intuition and judgment, which hardware engineers believed no technology could adequately provide. Trace logs used for debugging, for example, could highlight for hardware engineers what kind of problem occurred, where it occurred, and what ramifications it had. However, the trace logs could not tell the engineers what caused the problem (e.g., giving a variable a wrong name, mistakenly calculating the timing of an event, or putting the wrong data in a register). To narrow backward gaps by entering the testing feedback as direct input into a technology for modifying code far exceeded the analytic capabilities of technologies. Whereas hardware engineers believed that bridges built across forward gaps automated tasks that required routine or codifiable knowledge, they were certain that contending with feedback required skilled problem solving on the part of the engineer, resulting in no narrowing of backward gaps.

Unlike hardware engineers, structural engineers did not use the phrase "design flow" or any equivalent term. Structural engineers understood technology interdependence in the context of their work (that is to say, they recognized the issues of transfer and transformation as work products traveled among technologies), but, compared to hardware engineers, they managed this interdependence much less fastidiously. That is not to say that structural engineers never purchased new technologies to improve their flow, but, in general, they purchased new technologies absent the detailed probing, extended vendor interaction, and assessment witnessed in hardware engineering. Perhaps as a result, new technology purchases did little to streamline the flow. In a case described by a senior engineer, the modeling assumptions of a new application were not aligned with the assumptions that engineers routinely made. Consequently, although the application automated one task, using it required the extra step of transforming input data:

> We needed a good tool to do three-dimensional modeling. If you have a rectangular building with all the floors the same, then you ought to be able to build a model that puts in all the coordinates for you automatically, which is what this new software does. However, because some programmer assumed that your model is parallel to the y axis, we have to re-input all of the data. That is a ridiculous assumption to make. Programmers don't really know what assumptions make sense.

At least three factors prevented greater coordination among interdependent technologies in structural engineering. The first was the availability of multiple technologies for most tasks. As we noted, the low purchase cost of software and the continued validity of the domain knowledge that underpinned the algorithms embedded in software meant that, over time, numerous technologies existed for most tasks. Choosing which technology to use was often a matter of engineers' preference, with no consensus on which one was "best." The multiplicity of technologies resulted in little standardization of input and output formats, which meant that multiple interface problems had to be solved. For example, if four technologies were available for one task and only one technology for the follow-on task, four distinct gaps existed. If two technologies rather than one were available for the second task, the number of gaps jumped to eight. In this sense, although pooled technology interdependence generated no problems for structural engineers when they were choosing within a set of technologies for a task, it did contribute to larger problems associated with traversing gaps between technologies in sequential sets.

Why not simply ban certain technologies to reduce the magnitude of the interface problem? The answer to that question may be found in the second factor that impeded the narrowing of technology gaps in structural engineering: the belief among senior engineers that younger engineers' fondness for and reliance on computer analysis prevented them from gaining fundamental design knowledge. Such knowledge was important in structural engineering because software programs for testing and analysis could verify that a design was sound given its assumptions. However, they could not assess whether the assumptions themselves were appropriate. Senior engineers thus stressed the continued occupational value of verifying one's assumptions by traditional practices such as tracing load paths through drawings and using proven approximations to estimate forces rather than repeatedly running simulations until a "feasible"—but all too often unrealistic—solution was achieved. The comments of a senior engineer about an analysis software application whose built-in assumptions ran contrary to the physics of a particular building design illustrate the tension between computer analysis and traditional design practices:

> That software has several weird assumptions, including a fixed-base assumption. I reviewed a building where it had been used. I could tell just by looking at it that it was a poor design. They sent me reams of computer printouts from it, but all I needed to do was look at the drawing. There was a place on the drawing where a beam came in at a 45-degree angle, and that beam was huge because the engineer said it was taking all the load because it had a fixed base. But beside it was this other beam, and it was really skinny. And the thing is, the software kept telling the engineer to make the first beam bigger, and it kept taking more of the load, so it got bigger and bigger and this other one got really skinny. But that was all based on the assumption of the fixed base, and in reality it wasn't fixed at all! In fact, this thing was sitting on almost nothing, so all the load really had to go through that very skinny member. It all happened because some young kid out of college learned the software, and just did what the software told him to do.

Senior engineers reasoned that leaving wide forward gaps aided in the occupational training of junior engineers by cultivating the development of design knowledge. For this reason, senior engineers placed little emphasis on improving work efficiency by narrowing gaps. Similar logic explains why structural engineers left wide backward gaps. Like hardware engineers, they recognized that design modifications based on feedback required engineering judgment that occupational members deemed too important to lose to automation.

Liability concerns were a third factor that lessened the perceived importance of narrowing gaps in structural engineering. The engineer who submitted a building's design for a county permit was held responsible if the structure failed. Avoiding liability problems meant designing structures free of major errors. Unlike hardware engineers, structural engineers did not trust that software programs could adequately test for every identified type of error. In addition, structural engineers viewed each design as unique, with idiosyncratic design issues that required careful reflection and reexamination through each step of design and analysis. Consequently, structural engineers were reticent to narrow technology gaps, and in doing so abdicate control to the analysis applications, because they could ill afford to lose the discretion to apply their occupational knowledge and expertise.

These deterrents to narrowing gaps in this field help explain why structural engineers had limited interaction with individuals outside the firm who might aid them in bridge-building activities. Structural engineers occasionally did send feedback to vendors about technology problems, typically via forms on the vendors' websites. However, they did not engage in more proactive co-development or licensing activities. In addition, structural engineers neither attended conferences dedicated to new technologies nor communicated electronically with other members of the profession in external firms about technologies.

Overall, for structural engineers, plying a path among interdependent technologies was primarily a solitary, unassisted endeavor. Because senior engineers in particular viewed the navigation of wide gaps as beneficial for the cultivation of occupational design knowledge and as prudent in the face of liability concerns, there was little imperative to improve technology coordination by limiting the number of technologies that lined each gap in structural engineering: If easy gap traversal was not the goal, then all traversal options, being rather equally arduous, were viable, leaving no good reason to deny an engineer his preference. For similar reasons, engineers were not encouraged to build bridges across technology gaps.

Discussion

Implications for Theories of Organizing Work

At the broadest brush, interdependence among technologies looks quite similar to interdependence among people. When complex tasks are divided into subtasks to be completed by different technologies (people), the technologies (people) must integrate their component parts to finish the whole task and hence are interdependent (see Thompson 1967). The division of work among technologies (people) creates gaps; the completion of work requires integration across these gaps. In short, interdependence entails minding the gaps created in the division of work. Where technology interdependence departs from task interdependence is in the degree of coordination required to mind these gaps successfully.

In the case of interdependence among people, organizations enact routines, plans, and schedules in the hopes of minding the gaps via strong coordination (Malone and Crowston 1994). Communication is another main coordinating mechanism (March and Simon 1958). Beyond explicit mechanisms such as routines and communication, interdependent individuals may also implicitly coordinate work by anticipating the actions and needs of team members and adjusting their own actions accordingly (Rico et al. 2008). Heath and Staudenmayer (2000) argued that, despite this multiplicity of coordination mechanisms, individuals often neglect coordination across interfaces because they are too cognitively preoccupied with the act of partitioning the large task into component tasks (and with the individual components created by this partitioning) to pay sufficient attention to integrating the components back into a whole. Heath and Staudenmayer further claimed that individuals may neglect coordination when they inadequately communicate or insufficiently translate across interfaces. Neglecting coordination can be detrimental because research on interdependence among people is universal in its contention that successful completion of interdependent tasks requires coordination; the higher the interdependence is, the greater is the need for coordination (e.g., Ilgen 1999, Rico et al. 2008, Wageman 1995).

In the case of interdependence among technologies, efforts to mind the gaps typically focus on standardizing input and output, as in the case of exchange and file formats for digital technologies. Problems of coordination across interfaces, or what we have termed technology gaps, are frequently posed as problems of interoperability, which refers to the ability of technologies to share data with ease (March et al. 2000). In our study, hardware engineers coordinated technologies through bridge-building activities, such as writing scripts to automate the transfer of data from one technology to another. Coordination was also the goal when hardware engineers purchased new equipment that fit into the design flow. In contrast, structural engineers often neglected to coordinate their technologies, but not for the reasons outlined by Heath and Staudenmayer (2000). Rather, structural engineers purposefully eschewed high coordination because they believed that navigating difficult interfaces promoted the development of necessary occupational design knowledge.

In addition, liability concerns and the inability of technologies to thoroughly test building designs provided structural engineers with good reason to maintain difficult interfaces as opportunities for examination of models and reflection on their soundness. Thus, although engineering work in both occupations was divided across a range of technologies and had to be integrated for completion of the whole, coordination was much greater in one occupation than in the other. In short, greater coordination of technologies was not universally sought or necessary for successful completion of highly interdependent tasks.

Our findings have important implications for theories of organizing work. These findings suggest that the division of work among technologies can support larger occupational and organizational goals. Correct placement of technology gaps via this division can serve, for example, to enable quality inspection of the product, to meet training needs of individuals, to preserve occupational knowledge, and possibly even to maintain the status distinction between senior and junior engineers. One might infer then that incorrect placement would inhibit achievement of these goals. Longer segments of work could be assigned to a single technology whenever occupational and organizational goals did not require a break in the form of a technology gap. Overall, efforts to improve work productivity by bridging technology gaps without considering occupational and organizational goals could disrupt beneficial, albeit time-consuming, gap traversal strategies that meaningfully contribute to the development of the product or the workforce.

Implications for Theories of Social Dynamics of Work

Our findings also have implications for theories related to the social dynamics of work. Studies of interdependence among people have shown that interdependence is related to cooperation, learning, citizenship, helping, motivation, and satisfaction (Spriggs et al. 2000, Van der Vegt et al. 2003, Wageman 1995, Wageman and Baker 1997). For example, Wageman (1995) found that highly interdependent work groups exhibited high cooperation, helping, and mutual learning that were less apparent among low interdependence groups. In the case of interdependence among technologies, human effort is required to traverse gaps. Therefore, cooperation, helping, and other social dynamics in relation to technology interdependence can be viewed from the perspective of how individuals interact with respect to gap traversal. When hardware engineers built bridges to traverse gaps, they were engaged in cooperation and helping because bridges could be used by everyone (given that each engineer worked with the same suite of technologies, an outcome of how work was divided among *people*). Engineers saw bridge-building actions as helping even though the first person aided by the bridge was the builder himself. These results among hardware engineers suggest that, when coordination is the goal and when task interdependence is structured such that many individuals experience technology interdependence identically, high technology interdependence will be associated with high levels of helping until all gaps are bridged.

Among structural engineers, social dynamics were heavily influenced by occupational hierarchy, a situation that we did not observe among hardware engineers. Through their decisions to maintain multiple technologies and their development of norms that downplayed bridge-building efforts, senior structural engineers acting as managers shaped how junior engineers traversed technology gaps, forcing them to navigate rather than cross gaps. Senior engineers designed explicit policies around technology interdependence and coordination that were not aimed at managing technology most efficiently, for to do so would surely have involved streamlining the design flow and automating the transfer of work products across technology gaps. Rather, their decisions with respect to technology interdependence were aimed at managing engineers in a manner consistent with developmental goals and liability concerns that reflected occupational and industry concerns, respectively.

In the case of structural engineers, unlike that of hardware engineers, the most salient social dynamics appeared to be independent of the structure of task interdependence. That is to say, senior structural engineers maintained wide technology gaps even if, for example, work moved from one engineer to another at the same time it transferred from one technology to another. In structural engineering, a change in who does what with which technology would not alter the need to develop design knowledge or to examine models for their soundness. In contrast, in hardware engineering, the social dynamics appeared to be an outcome of the particular confluence of task and technology interdependence. Bridge-building activities conceivably would look different if hardware engineers handed off work to one another across technology gaps, because a bridge built from one's own technology to the one before it would only help oneself (enabling better transfer to one's tool), and a bridge built to the technology after it would only help the next engineer (enabling better transfer to that tool). Consequently, the value of each bridge would be localized, and helping would be less visible to the group as a whole. In short, if task and technology interdependence were more closely aligned in hardware engineering, the motivation for building bridges might change, so bridge-building activity might look different than it did with pooled task interdependence with sequential technology interdependence. Overall, across both occupations, it seems clear that technology interdependence was related to a particular set of social dynamics.

Looking Forward

Given that knowledge tasks are becoming increasingly data driven (Carlson 2001, Dodgson et al. 2007, Markus 2004) and technologies are often used to automate routine work processes (Henderson 1998, Stohr and Zhao 2001, Suchman 2007), might not all occupations eventually come to deal with technology gaps in a manner similar to the way hardware engineers deal with them? One way of answering this question is to first consider whether structural engineers might someday bridge technology gaps.

Far from being Luddites, structural engineers were early adopters of computerization beginning more than 50 years ago (Bédard 2006). With the introduction of the desktop computer in the 1980s, a growing market for applications provided engineers with many options. Today, calls for integration across these technologies certainly exist, but given the fragmentation of construction projects across architectural firms, engineering firms, and contractors, the calls are primarily aimed at interfirm communication and data sharing (e.g., Rosenman et al. 2007, Yang and Zhang 2006). Much of the ensuing discussion is thus ontological, centering on the construction of standard data and interface formats to facilitate sharing of models across similar tools in different firms. In other words, the focus is on, for example, allowing architects and engineers with different CAD tools to view the same geometric model, rather than on helping a structural engineer easily transfer her geometric model created in a CAD tool to her computational tool for analysis. Although some research efforts are aimed at sharing data more easily up and down the structural engineering design flow (e.g., Senescu et al. 2006), these efforts are not center stage. Given the imperatives imposed by occupational development and industry liability, it seems doubtful that senior structural engineers will deem technology gaps within their firm as problematic as gaps across their firm to architects and contractors. As a result, structural engineers are unlikely to streamline technologies within their firms as hardware engineers have.

More broadly, in knowledge occupations beyond engineering, we suspect that a number of factors may render the narrowing of wide gaps impractical, infeasible, or unwise. Cost is one such possibility: Firms may find that the cost of closing gaps is simply too high. Cost may prove a factor especially when in-house personnel lack the technical skills to manipulate technology interfaces, thus forcing the firm to consider contracting with external consultants, purchasing off-the-shelf applications, or hiring additional staff. Quite a few knowledge occupations are likely to reveal issues of liability that, as in the case of structural engineers, lend high value to wide gaps; lawyers, doctors, and stockbrokers seem among the likely candidates. Outsourcing or offshoring jobs at the task level requires wide technology gaps to allow transfer of the work between individuals (Leonardi and Bailey 2008). Similarly, task interdependencies across an organization's various functions and departments may correspond with wide technology gaps whose narrowing would impinge on existing successful work practices.

Enterprise resource planning (ERP) systems, or large-scale commercial software applications intended to integrate transaction-oriented work processes within a firm, arguably pose the greatest threat to the persistence of wide technology gaps in many knowledge occupations. ERP system implementation closes wide gaps by replacing independent computer applications, often unique to each function, with a set of interrelated programs in functional modules. The modules and their interfaces are standardized and permit little modification (Robey et al. 2002). Thus, when a firm implements an ERP system, it ignores the possibility that wide gaps may have served valuable purposes, often because the firm attributes their existence solely to the ad hoc acquisition of disjoint software applications over time. Boudreau and Robey's (2005) finding that ERP users developed workarounds even in the face of such a rigid technology suggests that these users may have sought ways to overcome the loss of control that accompanied the closing of wide gaps. Moreover, many organizations that adopt ERP systems find that they need to maintain some legacy programs and that linking those programs to the ERP system is difficult, meaning that narrow gaps are not easily achieved (Markus et al. 2000). Overall, research to date suggests that even large-scale integrated business systems are unlikely to remove all wide gaps in an organization.

Decreases in technology costs prompt a second, alternative future to one in which most wide gaps are narrowed, in that all occupations might come to look like structural engineering with its multiple technologies per task. We can begin to evaluate the likelihood of this possibility by first considering whether hardware engineers might someday have duplicate technologies for each task.

In hardware engineering, tasks continue to grow in complexity. Consider the task of verification: Because verification involves investigation of the total possible states of a chip's storage elements, the size of the verification problem is essentially Moore's law squared (MacMillen et al. 2000). To handle the increasing complexity of verification, hardware engineers have increased the level of abstraction at which they simulate models (Sangiovanni-Vincentelli 2003). In addition, new tools have been developed to formally check models (MacMillen et al. 2000). Because increasing complexity forces hardware engineers to simultaneously upgrade their current tools and add new tools to perform new subtasks, hardware engineers are unlikely to spend money on duplicate technologies for any step in the design flow even if technology prices begin to decrease.

More broadly, one can think of several other reasons some occupations might opt for a single technology per task. Perhaps, for example, an occupation may wish to ensure standardization of its product through use of a single technology whose processes and output are well understood. Smooth interfaces among existing technologies may similarly argue against the introduction of duplicate technologies for fear of introducing transfer problems that might impact productivity or quality; this possibility seems most likely in occupations whose members' skills do not include extensive technology manipulation. Firms may avoid duplication when implementation and maintenance costs of multiple technologies are prohibitive even though purchase costs are low. Finally, occupations that favor automation are apt to choose single technologies over multiple ones. In short, lower technology costs are unlikely by themselves to lead to a proliferation of technology options, and with them many wide gaps.

Although technology advances and decreases in cost are unlikely to alter our findings, there is still much to learn about technology interdependence, particularly in relation to task interdependence. In our study, technology interdependence was primarily sequential and task interdependence was primarily pooled. Because all engineers worked with the same suite of technologies on their distinct tasks, any change in the management of technology interdependence (e.g., the construction of a bridge) affected the work of all engineers (e.g., each engineer could traverse the bridged gap faster than before), but did not alter their task interdependence. The separation of task and technology interdependence in our study limits what we can say about their relationship. A simple thought experiment, however, suggests that if the two forms of interdependence are more tightly aligned, then changes in how technology interdependence is managed may have important implications for task interdependence and for occupational roles.

Consider the automotive engineering task of building and analyzing a digital simulation of a car crash. This task may be divided between two people such that the first person, a design engineer, is charged with designing the car and the second person, an analysis engineer, is charged with analyzing the design in a digital simulation. Sequential *task* interdependence is high: The analysis engineer cannot run the simulation absent the design of the car. Sequential *technology* interdependence also exists. The design engineer uses a CAD technology; the analysis engineer uses a computer-aided engineering (CAE) technology. Output from the CAD technology serves as input to the CAE technology, and vice versa in the case of feedback. If the technologies were to become highly coordinated (e.g., if bridges were built to cross the forward and backward gaps), task interdependence could change dramatically. High coordination could collapse the existing occupational division between design and analysis; if output from the CAD technology automatically fed into the CAE technology, the distinct roles of "design engineer" and "analysis engineer" might become obsolete. This thought experiment demonstrates that the level of coordination used to manage interdependent technologies has the potential to alter task interdependence and hence to reshape occupational roles. This experiment thus points to the potential of empirical research that explores the relationship between technology interdependence and task interdependence.

Future studies of forms of knowledge work may also discover yet more strategies that individuals use in dealing with technology gaps, adding to the typology that we created. New strategies may serve as behavioral clues to different occupational or organizational goals that are manifested in the coordination of technology. Such studies might also uncover situations in which individuals bridge backward gaps; the engineers in our study did not do so because traversal of these gaps required intuition and expertise that could not be given over to technologies. In other settings, traversing backward gaps may not require such skills.

More thorough inquiry is also required into the factors that shape technology interdependence; our study has uncovered several factors that appeared instrumental in the occupations we studied, but many more are likely to exist. Overall, the concepts of technology gaps and gap encounters that we put forward in this paper are but a first step in unraveling technology interdependence; future studies may develop other concepts that will deepen our understanding of this construct.

Acknowledgments

The authors thank Mahesh Bhatia, Fabrizio Ferraro, Menahem Gefen, Julie Gainsburg, and Lesley Sept for their assistance in data collection. The authors also thank the engineers for graciously allowing their work to be observed and for sharing their thoughts. This manuscript benefited immensely from the comments of three anonymous reviewers for *Organization Science* and Senior Editor Ann Majchrzak. Stephen Barley and two anonymous reviewers from the Academy of Management OCIS division provided helpful comments on earlier versions of this paper. This research was made possible by National Science Foundation (NSF) Grant IIS-0070468. The General Motors–Stanford Collaborative Research Laboratory provided additional funding.

Endnotes

[1] In some cases, such functions arguably represented technology gaps that had been managed by placing a standalone application within another program, but distinctions of this sort would have burdened our analysis efforts too heavily.

[2] A kip is a kilo-pound; one kip equals 1,000 pounds.

References

Bachrach, D. G., B. C. Powell, E. Bendoly, R. G. Richev. 2006. Organizational citizenship behavior and performance evaluations: Exploring the impact of task interdependence. *J. Appl. Psych.* **91**(1) 193–201.

Barley, S. R. 2005. What we know (and mostly don't know) about technical work. S. Ackroyd, R. Batt, P. Thompson, P. S. Tolbert, eds. *The Oxford Handbook of Work and Organization*. Oxford University Press, Oxford, UK, 376–403.

Bédard, C. 2006. On the adoption of computing and IT by industry: The case for integration in early building design. I. F. C. Smith, ed. *Intelligent Computing in Engineering and Architecture. Lecture Notes in Computer Science*, Vol. 4200. Springer, Heidelberg, Germany, 62–73.

Boland, Jr., R. J., K. Lyytinen, Y. Yoo. 2007. Wakes of innovation in project networks: The case of digital 3-D representations in architecture, engineering, and construction. *Organ. Sci.* **18**(4) 631–647.

Boudreau, M.-C., D. Robey. 2005. Enacting integrated information technology: A human agency perspective. *Organ. Sci.* **16**(1) 3–18.

Campbell-Kelly, M. 2003. *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry*. MIT Press, Cambridge, MA.

Carlson, P. A. 2001. Information technology and organizational change. *J. Technical Writing Comm.* **31**(1) 77–95.

Carrillo, J. E., C. Gaimon. 2000. Improving manufacturing performance through process change and knowledge creation. *Management Sci.* **46**(2) 265–288.

de Jong, S. B., G. S. Van der Vegt, E. Molleman. 2007. The relationships among asymmetry in task dependence, perceived helping behavior, and trust. *J. Appl. Psych.* **92**(6) 1625–1637.

Dodgson, M., D. M. Gann, A. Salter. 2007. "In case of fire, please use the elevator": Simulation technology and organization in fire engineering. *Organ. Sci.* **18**(5) 849–864.

Downey, G. L., A. Donovan, T. J. Elliott. 1989. The invisible engineer: How engineering ceased to be a problem in science and technology studies. *Knowledge Soc.* **8** 189–216.

Ecker, K. H., J. N. D. Gupta. 2005. Scheduling tasks on a flexible manufacturing machine to minimize tool change delays. *Eur. J. Oper. Res.* **164**(3) 627–638.

Espinosa, J. A., J. Lerch, R. Kraut. 2004. Explicit vs. implicit coordination mechanisms and task dependencies: One size does not fit all. E. Salas, M. Fiore, eds. *Team Cognition: Understanding the Factors That Drive Process and Performance*. APA Books, Washington, DC, 107–129.

Faraj, S., L. Sproull. 2000. Coordinating expertise in software development teams. *Management Sci.* **46**(12) 1554–1568.

Glaser, B. 1978. *Theoretical Sensitivity*. The Sociological Press, Mill Valley, CA.

Gray, A. E., A. Seidmann, K. E. Stecke. 1993. A synthesis of decision models for tool management in automated manufacturing. *Management Sci.* **39**(5) 549–567.

Guzzo, R. A., G. P. Shea. 1992. Group performance and intergroup relations in organizations. M. D. Dunnette, L. M. Hough, eds. *Handbook of Industrial and Organizational Psychology*, 2nd ed. Consulting Psychologists Press, Palo Alto, CA, 269–313.

Heath, C., N. Staudenmayer. 2000. Coordinating neglect: How lay theories of organizing complicate coordination in organizations. B. M. Staw, R. I. Sutton, eds. *Research in Organizational Behavior*. JAI Press, Greenwich, CT, 153–191.

Henderson, K. 1998. The role of material objects in the design process: A comparison of two design cultures and how they contend with automation. *Sci., Technology Human Values* **23**(2) 139–174.

Ilgen, D. R. 1999. Teams embedded in organizations: Some implications. *Amer. Psychologist* **54**(2) 129–139.

Langfred, C. W. 2007. The downside of self-management: A longitudinal study of the effects of conflict on trust, autonomy and task interdependence in self-managing teams. *Acad. Management J.* **50**(4) 885–900.

Leonardi, P. M., D. E. Bailey. 2008. Transformational technologies and the creation of new work practices: Making implicit knowledge explicit in task-based offshoring. *MIS Quart.* **32**(2) 411–436.

Linden, R. C., S. J. Wayne, L. K. Bradway. 1997. Task interdependence as a moderator of the relation between group control and performance. *Human Relations* **50**(2) 169–181.

MacMillen, D., M. Butts, R. Composano, D. Hill, T. W. Williams. 2000. An industrial view of electronic design automation. *IEEE Trans. Comput.-Aided Design Integrated Circuits Systems* **19**(12) 1428–1448.

Malone, T. W., K. Crowston. 1994. The interdisciplinary study of coordination. *ACM Comput. Surveys* **26**(1) 87–119.

March, J. G., H. Simon. 1958. *Organizations*. John Wiley & Sons, New York.

March, S., A. Hevner, S. Ram. 2000. Research commentary: An agenda for information technology research in heterogeneous and distributed environments. *Inform. Systems Res.* **11**(4) 327–341.

Markus, M. L. 2004. Technochange management: Using IT to drive organizational change. *J. Inform. Tech.* **19**(1) 4–20.

Markus, M. L., S. Axline, D. Petrie, S. C. Tanis. 2000. Learning from adopters' experiences with ERP: Problems encountered and success achieved. *J. Inform. Tech.* **15**(4) 245–265.

Mohr, L. B. 1971. Organizational technology and organizational structure. *Admin. Sci. Quart.* **16**(4) 444–459.

Morisi, T. L. 1996. Commercial banking transformed by computer technology. *Monthly Labor Rev.* **119**(8) 30–36.

Naveh, E., M. Erez. 2004. Innovation and attention to detail in the quality improvement paradigm. *Management Sci.* **50**(11) 1576–1586.

Rico, R., M. Sanchez-Manzanares, F. Gil, C. Gibson. 2008. Team implicit coordination processes: A team knowledge-based approach. *Acad. Management Rev.* **33**(1) 163–184.

Robey, D., J. W. Ross, M.-C. Boudreau. 2002. Learning to implement enterprise systems: An exploratory study of the dialectics of change. *J. Management Inform. Systems* **19**(1) 17–46.

Rosenman, M. A., G. Smith, M. L. Maher, L. Ding, D. Marchant. 2007. Multidisciplinary collaborative design in virtual environments. *Automation Construction* **16**(1) 37–44.

Sangiovanni-Vincentelli, A. 2003. The tides of EDA. *IEEE Design Test Comput.* **20**(6) 59–75.

Senescu, R., R. Mole, A. Fresquez. 2006. A case study in structural drafting, analysis and design using an integrated intelligent model. *Joint Internat. Conf. Comput. Decision Making in Civil Building Engineering, Montreal*, 1797–1806.

Sharma, R., P. Yetton. 2003. The contingent effects of management support and task interdependence on successful information systems implementation. *MIS Quart.* **27**(4) 533–556.

Shea, G. P., R. A. Guzzo. 1987. Groups as human resources. G. R. Ferris, K. M. Rowlands, eds. *Research in Personnel and Human Resources Management*. JAI Press, Greenwich, CT, 323–367.

Sinreich, D., B. D. Nelkenbaum. 2006. Determining production sequences for single-stage multifunctional machining systems based on the tradeoff between fixture cost, re-fixturing and tool replenishment. *IIE Trans.* **38**(10) 813–828.

Spriggs, C. A., P. R. Jackson, S. K. Parker. 2000. Production teamworking: The importance of interdependence and autonomy for employee strain and satisfaction. *Human Relations* **53**(11) 1519–1542.

Staum, J. 2001. Simulation in financial engineering. B. A. Peters, J. S. Smith, D. J. Medeiros, M. W. Rohrer, eds. *Proc. 2001 Winter Simulation Conf.*, IEEE Press, Los Alamitos, CA, 123–133.

Stohr, E. A., J. L. Zhao. 2001. Workflow automation: Overview and research issues. *Inform. Systems Frontiers* **3**(3) 281–296.

Strauss, A., J. Corbin. 1998. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*, 2nd ed. Sage, Thousand Oaks, CA.

Streufert, S., U. Satish, P. Barach. 2001. Improving medical care: The use of simulation technology. *Simulation Gaming* **32**(2) 164–174.

Suchman, L. 2007. *Human–Machine Reconfigurations: Plans and Situated Actions*. Cambridge University Press, Cambridge, MA.

Thompson, J. D. 1967. *Organizations in Action: Social Science Bases of Administrative Theory*. McGraw-Hill, New York.

Van der Vegt, G. S., O. Janssen. 2003. Joint impact of interdependence and group diversity on innovation. *J. Management* **29**(5) 729–751.

Van der Vegt, G. S., E. Van de Vliert. 2005. Effects of perceived skill dissimilarity and task interdependence on helping in work teams. *J. Management* **31**(1) 73–89.

Van der Vegt, G. S., E. Van de Vliert, A. Oosterhof. 2003. Informational dissimilarity and organizational citizenship behavior: The role of intrateam interdependence and team identification. *Acad. Management J.* **46**(6) 715–727.

Wageman, R. 1995. Interdependence and group effectiveness. *Admin. Sci. Quart.* **40**(1) 145–180.

Wageman, R., G. Baker. 1997. Incentives and cooperation: The joint effects of task and reward interdependence on group performance. *J. Organ. Behav.* **18**(2) 139–158.

Yang, Q. Z., Y. Zhang. 2006. Semantic interoperability in building design: Methods and tools. *Comput.-Aided Design* **38**(10) 1099–1112.

Zantek, P. F., G. P. Wright, R. D. Plante. 2002. Process and product improvement in manufacturing systems with correlated stages. *Management Sci.* **48**(5) 591–606.

Zuboff, S. 1988. *In the Age of the Smart Machine: The Future of Work and Power*. Basic Books, New York.