Knowledge Management Research & Practice (2004) 2, 169–183
© 2004 Palgrave Macmillan Ltd. All rights reserved 1477–8238/04 $30.00
www.palgrave-journals.com/kmrp

Agent-based modeling of knowledge dynamics

Mark E Nissen[1]
Raymond E Levitt[2]

[1]Naval Postgraduate School, Monterey, CA, USA; [2]Stanford University, Stanford, CA, USA

Correspondence: Mark E. Nissen, Naval Postgraduate School, 555 Dyer Road, Code GB/Ni, Monterey, CA 93943-5000, USA
Tel: +1 831 656 3570; Fax: +1 831 656 3407;
E-mail: MNissen@nps.edu

Abstract
Knowledge is distributed unevenly through most enterprises. Hence, flows of knowledge (e.g., across time, people, locations, organizations) are critical to organizational efficacy and performance under a knowledge-based view of the firm. However, supported principally by narrative textual theory in the emerging knowledge management (KM) field, the researcher has difficulty describing how different kinds of knowledge will flow through various parts of an organization. This causes difficulty also for predicting the effects of alternate approaches to dispersing knowledge that 'clumps' in various areas. This problem is also manifest for the KM professional, who lacks clear theory or tools to anticipate how any particular information technology or other managerial intervention may enhance or impede specific knowledge flows in the enterprise. In this expository article, we build upon a steady stream of research in computational organization theory to develop agent-based models of knowledge dynamics. This work draws from emerging theory for multi-dimensional representation of the knowledge-flow phenomenon, which enables the dynamics of enterprise knowledge flows to be formalized and emulated through computational models. This approach provides the means for knowledge-flow processes to be visualized and analyzed in new ways. Computational experimentation enables the performance of many alternate process designs and technological interventions to be compared through examination of dynamic models, before committing to a specific approach in practice. We illustrate this research method and modeling environment through semi-formal representation and agent-based emulation of several knowledge-flow processes from the domain of software development. We also outline key directions for the new kinds of KM research and practice elucidated by this work.
Knowledge Management Research & Practice (2004) **2,** 169–183.
doi:10.1057/palgrave.kmrp.8500039

Keywords: computational organization theory; knowledge flows; knowledge representation; organizational learning; simulation; software

Introduction

Knowledge represents a critical resource in the modern enterprise – so critical that it is conceptualized as central to competitive advantage in a knowledge-based view of the firm (Grant, 1996; Spender, 1996; Cole, 1998). But knowledge is not distributed evenly through the enterprise. Capitalizing on this resource for enterprise performance depends upon its rapid and efficient transfer from one organization, location, person or time of application to another. From a technological perspective, such dynamic dependence points immediately to the design of information systems (IS) – along with corresponding organization, work and process redesigns (Leavitt, 1965; Davenport, 1993) – to enhance knowledge flows. But knowledge is distinct from information (e.g., it enables action; see Nonaka, 1994; Davenport *et al*., 1998; Teece, 1998), and few extant *information*

Received: 12 December 2003
Accepted: 15 September 2004

systems even address *knowledge* as the focus or object of flow (Nissen, 1999).

In the context of research, the emerging knowledge management (KM) field does not have the benefit of strong theory on knowledge flows. As Alavi & Leidner (2001, p. 126) note, there exist 'large gaps in the body of knowledge in this area.' Limited by our present state of knowledge, the researcher has difficulty describing how different kinds of knowledge will flow through various parts of an organization. This causes difficulty also for predicting the effects of alternate approaches to dispersing knowledge that 'clumps' in various areas. For instance, we have a number of models that describe various aspects of knowledge flows (e.g., Von Hippel, 1994; Dixon, 2000; Nissen *et al*., 2000; Schultze & Boland, 2000; Szulanski, 2000; Augier *et al*., 2001; King & Ko, 2001; O'Leary, 2001; Swap *et al*., 2001). But few such models address the dynamics of knowledge as it flows.

Nonaka (1994) describes a 'spiral' of dynamic interaction between tacit and explicit knowledge, which comprises a dynamic theory of knowledge creation. But even this dynamic model does not address explicitly the central variable *time*. Nissen (2002) builds upon Nonaka and others to develop a phenomenological model of knowledge dynamics. This latter model makes flow time explicit, and it supports a multidimensional representational framework for analysis and visualization of diverse knowledge-flow patterns. However, even this time-dependent characterization of knowledge-flow dynamics remains static in terms of its representational model. This is much like trying to visualize three-dimensional motion through a static picture (e.g., camera snapshot).

Further, all of the extant theoretical models above are constructed principally from natural language textual narrative, which is inherently ambiguous and informal. Different researchers develop different interpretations of what a theory says, which magnifies differences in terms of what such theory means. This confounds theory development and impedes theory testing. The problem is also manifest for the KM professional. Ambiguous theory is limited in its utility to inform practice. The professional further lacks tools to anticipate how any particular information technology or other managerial intervention may enhance or impede specific knowledge flows in the enterprise. Without clear theory or predictive tools, the manager is limited generally to reliance upon intuition, imitation or trial and error, techniques that do not enjoy the benefits of cumulative knowledge accretion.

The research described in this article continues the modeling work from above by addressing the dynamics of knowledge flows. Drawing upon current advances in computational organization theory, it develops agent-based models of knowledge dynamics. Through commitment to a semi-formal representation of the knowledge-flow phenomenon, the resulting models are considerably less ambiguous and more precise than even the richest of those based principally upon natural language textual description. Agent-based models also enable the execution and performance of diverse knowledge work processes to be emulated for analysis and comparison – including the ability to stop, re-examine and replay repeatedly the action of knowledge flows – and the approach is broadly generalizable. This represents a new contribution to KM research. Agent-based models can further be employed as analytical tools to assess computationally the relative performance of alternate technologies and managerial interventions to enhance knowledge flows – before committing to implementation in the physical organization (e.g., via trial and error). This represents a new contribution to KM practice.

In this article, we take an expository approach to describing agent-based modeling of knowledge dynamics. Such modeling builds upon substantial prior research (e.g., static model integration, agent-based model representation). We summarize background research in the first half of the article and cite it liberally to guide the reader for further reference. Such modeling also enables a novel computational approach to KM research and practice. We illustrate the associated dynamic knowledge-flow behaviors through examples from the domain of software development. The article concludes by summarizing key directions for the new kinds of KM research and practice elucidated by this work.

Static model integration

The Spiral Model described by Nonaka (1994) serves as the cornerstone of model integration in this section. Figure 1 delineates the interaction between epistemological and ontological dimensions used by Nonaka as the principal means for describing knowledge as it flows through the enterprise. Four enterprise processes (and epistemological conversions) characterize this flow: socialization (tacit to tacit), externalization (tacit to explicit), combination (explicit to explicit), and internalization (explicit to tacit). The related *trigger* concept (Nonaka *et al*., 1996) is also integrated into the figure to

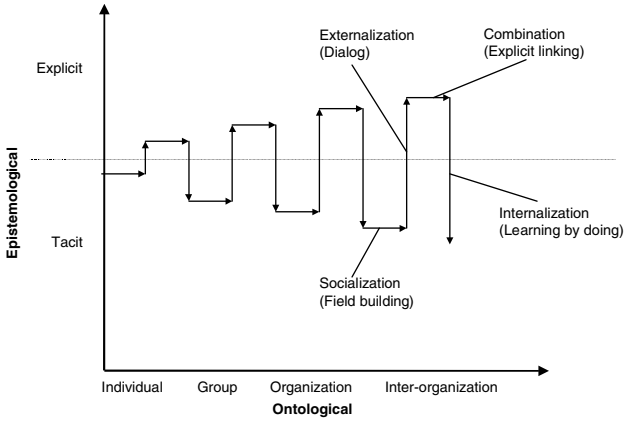

Figure 1 Spiral Model (adapted from Nonaka, 1994; Nonaka *et al*., 1996).

show where each knowledge-conversion process is 'induced' (p 842) by one of four triggers: field building, dialog, linking explicit knowledge, and learning by doing, respectively.

Briefly, socialization denotes members of a team sharing experiences and perspectives, much as one anticipates through tightly knit workgroups and communities of practice; in terms of the trigger, a 'field' of interaction is seen as facilitating this sharing. Externalization denotes the use of metaphors through dialog that leads to articulation of tacit knowledge and its subsequent formalization to make it concrete and explicit; such dialog or what Nonaka *et al*. refer to as 'collective reflection' (p 842) is described as inducing externalization. Combination denotes coordination between different groups in the organization – along with documentation of existing knowledge – to link and combine new intra-team concepts with other explicit knowledge in the enterprise. Internalization denotes diverse members in the organization applying the combined knowledge from above – often through trial and error – and in turn translating such knowledge into tacit form at the organization level (e.g., through work practices and routines); the term *learning by doing* is used to describe the trigger for knowledge internalization. As suggested by the repeating pattern delineated in the figure, such interaction between 'triggers' and conversions enables a continuous 'spiral' of knowledge.

We extend this Spiral Model by drawing from Nissen *et al*. (2000), who develop their Life Cycle Model by amalgamating several individual works (e.g., Davenport & Prusak, 1998; Despres & Chauvel, 1999; Gartner Group, 1998; Nissen, 1999). This amalgamated model describes a continuous cycle with six phases of knowledge flowing through the enterprise: (1) creation, (2) organization, (3) formalization, (4) distribution, (5) application, and (6) evolution. Briefly, the creation phase begins the life cycle, as new knowledge is generated within an enterprise; similar terms from other models include *capture* and *acquire*. The second phase pertains to the organization, mapping or bundling of knowledge, often employing systems such as taxonomies, ontologies, and repositories. Phase 3 addresses mechanisms for making knowledge formal or explicit; similar terms from other models include *store* and *codify*. The fourth phase concerns the ability to share or distribute knowledge in the enterprise; this also includes terms such as *transfer* and *access*. Knowledge use and application for problem solving or decision making in the organization constitutes Phase 5. A sixth phase is included to cover knowledge refinement and evolution, which reflects organizational learning – and thus a return to knowledge creation – through time. It is important to note, as in the familiar life cycle models used in IS design (e.g., System Development Life Cycle or SDLC), progression through the various phases of this Life Cycle Model is generally iterative, and it involves feedback loops between stages; that is, all steps need not be taken in order, and the flow through this life cycle is not necessarily unidirectional.

Integrating the concepts above, the resulting model can be represented using three dimensions: *epistemological*, *ontological*, and *life cycle*. Clearly, many additional dimensions could also be integrated into this model (e.g., declarative and procedural, see Nolan Norton (1998); practical and theoretical, see Spender (1996); know-what and know-how, see Ryle (1958); causal, conditional and relational, see Zack (1998); embodied, encoded, embrained, embedded and procedural, see Venzin *et al*., 1998). However, using three dimensions strikes a balance between descriptiveness and parsimony, and the three dimensions selected for this model all derive from research focused specifically on knowledge flows. We use this three-dimensional representation to characterize in new ways the complex interactions between knowledge in alternate states as it flows through the enterprise. Yet we also preserve the descriptive and explanatory abilities of the individual models that underlie (and are subsumed by) this integrative work.

The three-dimensional representation also enables us to visualize a diversity of enterprise knowledge flows in terms of a vector space and to plot dynamic trajectories for each flow. Drawing from Nissen (2002), three notional knowledge-flow trajectories are plotted in Figure 2 for illustration. For instance, the simple linear flow labeled 'P&P' depicts the manner in which most large enterprises inform and attempt to acculturate employees through the use of policies and procedures: explicit documents and guidelines that individuals in the organization are expected to memorize, refer to and observe. As another instance, the cyclical flow of knowledge through a life cycle (labeled 'KMLC') reflects a more complex dynamic than its simple linear counterpart – similar to that expected when an individual participates in a community of practice, for instance. The 'spiral' dynamic from above can also be delineated in this space by the curvilinear vector sequence **S–E–C–I** (i.e., corresponding to the processes of socialization, externalization,

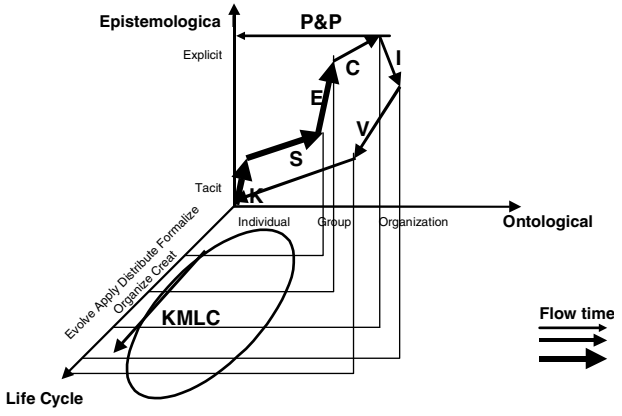

Figure 2 Notional knowledge-flow trajectories (adapted from Nissen, 2002).

combination, and internalization, respectively). Further, drawing from the Life Cycle Model, we can append processes to represent knowledge creation and evolution (labeled as vectors **K** and **V**, respectively), which we link together in turn to depict a serpentine flow that spans the ranges of all three dimensions.

Notice in this representation that we can depict explicitly flow times of various magnitudes by adjusting proportionately the thickness of arrows used to represent each knowledge-flow vector. This enables us to differentiate graphically between specific flow processes and states associated with 'sticky' and 'fluid' knowledge (see Von Hippel, 1994; Szulanski, 2000), for example. This enables us also to diagnose organizational work process areas in which knowledge remains 'clumped' and to identify managerial interventions appropriate to disperse such knowledge. Notice also that different knowledge flows delineate discernable patterns in this vector-space representation (e.g., lines, cycles, spirals). Clearly, a great many flows and patterns can be depicted in this manner.

Agent-based model representation

Despite the richness of our multidimensional representation above, the representation itself remains static; that is, although we use it to describe the dynamics of knowledge-flow processes, the representation is limited to a static combination of natural language texts and diagrams. In this section, we draw upon current advances in computational organization theory (see Carley & Prietula, 1994) to describe an agent-based representational environment used for semi-formal organizational modeling. We employ such environment first in this section to describe a computational model of software development workflow processes. This sets the stage for subsequent comparison with knowledge-flow models further below.

Virtual design team research

The Virtual Design Team (VDT) Research Program (VDT, 2004) reflects the planned accumulation of collaborative research over two decades to develop rich theory-based models of organizational processes. Using an agent-based representation (Cohen, 1992; Kunz *et al.*, 1998), micro-level organizational behaviors have been researched and formalized to reflect well-accepted organization theory (Levitt *et al.*, 1999). Extensive empirical validation projects (e.g., Christiansen, 1993; Thomsen, 1998) have demonstrated representational fidelity and have shown how the qualitative behaviors of VDT computational models correspond closely with a diversity of enterprise processes in practice.

The VDT research program continues with the goal of developing new micro-organization theory and embedding it in software tools that can be used to design organizations in the same way that engineers design bridges, semiconductors or airplanes: through computational modeling, analysis, and evaluation of multiple alternate prototype systems. Clearly, this represents a significant challenge in the domain of organizations. Micro-theory and analysis tools for designing bridges and airplanes rest on well-understood principles of physics (e.g., involving continuous numerical variables, describing materials whose properties are relatively easy to measure and calibrate), and analysis of such physical systems yields easily to differential equations and precise numerical computing.

In contrast, theories describing the behavior of organizations are characterized by nominal and ordinal variables, with poor measurement reproducibility, and verbal descriptions reflecting significant ambiguity. Unlike the mathematically representable and analyzable micro-behaviors of physical systems, the dynamics of organizations are: influenced by a variety of social, technical, and cultural factors; difficult to verify experimentally; and not amenable to numerical representation, mathematical analysis, or precise measurement. Moreover, quite distinct from physical systems, people and social interactions – not molecules and physical forces – drive the behavior of organizations; hence such behaviors are fundamentally non-deterministic and difficult to predict at the individual level. People, organizations, and business processes are qualitatively different than bridges, semiconductors and airplanes are, and it is irrational to expect the former to ever be as understandable, analyzable, or predictable as the latter. This represents a fundamental limitation of the approach.

Within the constraints of this limitation, however, we can still take great strides beyond relying upon informal and ambiguous, natural language textual description of organizational behavior (e.g., the bulk of extant theory). For instance, the domain of organization theory is imbued with a rich, time-tested collection of micro-theories that lend themselves to qualitative representation and analysis. Examples include Galbraith's (1977) information processing abstraction, March & Simon's (1958) bounded rationality assumption, and Thompson's (1967) task interdependence contingencies. Drawing from this theory base, we employ symbolic (i.e., non-numeric) representation and reasoning techniques from established research on artificial intelligence to develop computational models of theoretical phenomena. Once formalized through a computational model, the symbolic representation is 'executable,' meaning it can emulate the dynamics of organizational behaviors.

Even though the representation is qualitative (e.g., lacking the precision offered by numerical models), through commitment to computational modeling, it becomes semi-formal (e.g., people viewing the model can agree on what it describes), reliable (e.g., the same sets of organizational conditions and environmental factors generate the same sets of behaviors), and explicit (e.g., much ambiguity inherent in natural language is obviated). Particularly, when used *in conjunction with* the descriptive natural language theory of our extant literature, this represents a substantial advance. Further, once a model has been validated to emulate accurately the

qualitative behaviors of the field organization it represents, it can be used to examine a multitude of cases (e.g., many more and diverse than observable in practice) under controlled conditions (e.g., repeating the same events multiple times, manipulating only one or a few variables at a time through repeated trials, stopping the action for interpretation). This alone offers great promise in terms of theory development and testing.

Additionally, although organizations are inherently less understandable, analyzable, and predictable than physical systems are, and the behavior of people is nondeterministic and difficult to model at the individual level, it is known well that individual differences tend to average out when aggregated cross-sectionally and/or longitudinally. Thus, when modeling aggregations of people in the organizational context (e.g., work groups, departments, firms), one can augment the kind of symbolic model from above with certain aspects of numerical representation. For instance, the distribution of skill levels in an organization can be approximated, in aggregate, by a Bell curve; the probability of a given task incurring exceptions and requiring rework can be specified, organization wide, by a distribution; and the unpredictable attention of a worker to any particular activity or event (e.g., new work task, communication, request for assistance) can be modeled, stochastically, to approximate collective behavior. As another instance, specific organizational behaviors can be simulated hundreds of times – such as through Monte–Carlo techniques – to gain insight into which results are common and expected *vs* those that are rare and exceptional.

Of course, applying numerical simulation techniques to organizations is nothing new (e.g., see Law & Kelton, 1991). However, this approach enables us to *integrate* the kinds of dynamic, qualitative behaviors emulated by symbolic models with quantitative aggregate dynamics generated through discrete-event simulation. It is through such integration of qualitative and quantitative models – bolstered by strong reliance upon well-established theory and commitment to empirical validation – that our approach diverges most from extant research methods and offers new insight into the dynamics of organizational behavior.

VDT modeling environment

Here we provide a brief overview of the VDT modeling environment. The development and evolution of VDT has been described in considerable detail elsewhere (e.g., Cohen, 1992; Christiansen, 1993; Jin & Levitt, 1996; Thomsen, 1998; Kunz *et al*., 1998; Levitt *et al*., 1999; Nogueira, 2000; VDT, 2004), so we do not repeat such discussion here. The VDT modeling environment has been developed directly from Galbraith's information processing view of organizations. This information processing view has two key implications (Jin & Levitt, 1996). The first is ontological: we model knowledge work through interactions of *tasks* to be performed, *actors* communicating with one another and performing tasks, and an *organization structure* that defines actors' roles and that constrains their behaviors. In essence, this amounts to overlaying the task structure on the organization structure and to developing computational agents with various capabilities to emulate the behaviors of organizational actors performing work.

Figure 3 illustrates this view of tasks, actors, and organization structure. As suggested by the figure, we model the organization structure as a network of reporting relations, which can capture micro-behaviors such as managerial attention, span of control and empowerment. We represent the task structure as a separate network of activities, which can capture organizational attributes such as expected duration, complexity, and required skills. Within the organization structure, we further model various *roles* (e.g., marketing analyst, design engineer, manager), which can capture organizational attributes such as skills possessed, level of experience, and task familiarity. Within the task structure, we further model various sequencing constraints, interdependencies and quality/rework loops, which can capture considerable variety in terms of how knowledge work is organized and performed.

As suggested also by the figure, each actor within the intertwined organization and task structures has a queue of information tasks to be performed (e.g., assigned work activities, messages from other actors, meetings to attend) and a queue of information outputs (e.g., completed work products, communications to other actors, requests for assistance). Each actor processes such tasks according to how well the actor's skill set matches those required for a given activity, the relative priority of the task, the actor's work backlog (i.e., queue length), and how many interruptions divert the actor's attention from the task at hand. Collective task performance is constrained further by the number of individual actors assigned to each task, the magnitude of the task, and both scheduled (e.g., work breaks, ends of shifts, weekends, and holidays) and unscheduled (e.g., awaiting managerial decisions, awaiting work or information inputs from others, performing rework) downtime.

The second implication is computational: both primary work (e.g., planning, design, management) and coordination work (e.g., group tasks, meetings, joint problem

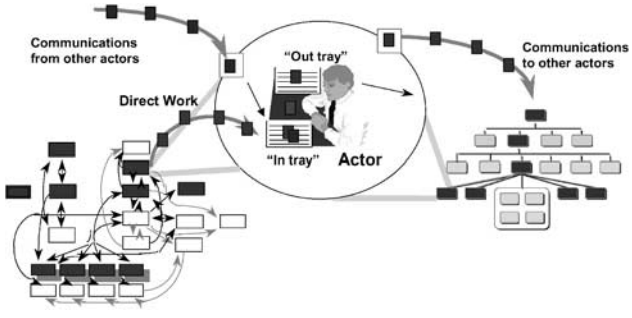

Figure 3 VDT information processing view of knowledge work (adapted from Jin & Levitt, 1996).

solving) are modeled in terms of *work volume*. This construct is used to represent a unit of work (e.g., associated with a task, a meeting, a communication) within the task structure. In addition to symbolic execution of VDT models (e.g., qualitatively assessing skill mismatches, task-concurrency difficulties, decentralization effects) through micro-behaviors derived from organization theory, the discrete-event simulation engine enables (virtual) process performance to be assessed (e.g., quantitatively projecting task duration, cost, rework, process quality).

Clearly, quantitative simulation places additional burden on the modeler in terms of validating the representation of a knowledge-work process, which generally requires fieldwork to study an organization in action. The VDT modeling environment benefits from extensive fieldwork in many diverse enterprise domains (e.g., power plant construction and offshore drilling, see Christiansen (1993); aerospace, see Thomsen (1998); software development, see Nogueira (2000); healthcare, see Cheng & Levitt (2001); others). Through the process of 'backcasting' – predicting known organizational outcomes using only information that was available at the beginning of a project – VDT models of operational enterprises in practice have demonstrated dozens of times that emulated organizational behaviors and results correspond qualitatively and quantitatively to their operational counterparts in the field (Kunz *et al.*, 1998).

Viewing VDT as a validated model of project-oriented knowledge work, researchers have begun to use this dynamic modeling environment as a 'virtual organizational testbench' to explore a variety of organizational questions, such as effects of distance on performance (Wong & Burton, 2000), or to replicate classic empirical findings (Carroll & Burton, 2000). Thus, the VDT modeling environment has been validated repeatedly and longitudinally as representative of both organization theory and enterprises in practice. This gives us considerable confidence in its results. However, because of its *information* processing view of the organization, the VDT modeling environment was not designed specifically to represent processes associated with flows of *knowledge* through an enterprise.

Baseline computational software development model

Here we employ the VDT modeling environment to represent work processes associated with software development. With its output the product of collaboration between people and computers, software development represents a relatively pure form of knowledge work, and the associated processes and technologies represent the focus of many KM programs in practice. This domain also helps to elucidate our multi-dimensional knowledge-flow representation from above, and it highlights both differences and linkages between flows of work in the enterprise and the corresponding flows of knowledge.

Building upon prior research in this domain (e.g., Nogueira, 2000), we specify first the VDT modeling environment in terms of agents that reflect behaviors appropriate for software development *workflows*. For instance, actor agents in VDT perform work tasks according to various *skill levels* (e.g., 'low' for novice, 'medium' for journeyman, 'high' for master), which affect several qualitative behaviors (e.g., relative task efforts, quality levels, communication requirements) through rules, frames, and objects embedded within the agent-based emulation environment. Empirical field research indicates skill level exerts a greater influence on such behaviors in the software domain than it does on behaviors associated with many other types of work (e.g., construction, manufacturing, service). We incorporate the benefits of such empirical relations into the model by selecting the *behavior file* appropriate for software development projects. Such behavior file represents a standard VDT element, which has been validated empirically, repeatedly, and longitudinally. VDT interprets such selection to fire rules and instantiate objects appropriate to emulate software development behaviors.

As another instance, in many work domains managers tend to be relatively more concerned about product quality than lower-level workers are. But empirical research of software processes indicates the opposite tends to hold; that is, software line and product managers tend to push more vigorously to meet schedule deadlines (e.g., at the expense of quality) than developers do. Thus the effects of different levels of hierarchy and organization structures vary between software processes and work in other domains. Our VDT model specification in terms of a software project causes the emulation to reflect such empirically validated behaviors.

We use a relatively simple work process for exposition of the software development model. Although the process is relatively simple, however, it does not represent a toy problem lacking real-world applicability or scalability. Rather, the nature of the work and behaviors of our agent-based emulation model reflect those of software development processes across a wide range of organizations, technologies, and projects. An overview of the VDT model for software development is illustrated through the screenshot presented in Figure 4. In this example, the organization structure is comprised of three elements: (1) a knowledge worker (i.e., person icon at top of diagram labeled 'S/W arch') with skills in software architecture development, (2) a team of (ten) software engineers (i.e., person icon at top of diagram labeled 'S/W engr team1') with skills in software analysis, design, and programming, and (3) a project lead (i.e., person icon at top of diagram labeled 'S/W PM') with skills in project planning, supervision, and software development. The task structure is comprised of two work elements: (1) software architecture (i.e., rectangular icon at middle of diagram labeled 'S/W arch'), and (2) software engineering (i.e., rectangular icon at middle of diagram labeled 'S/W engr'); the four milestones shown (i.e., project start,

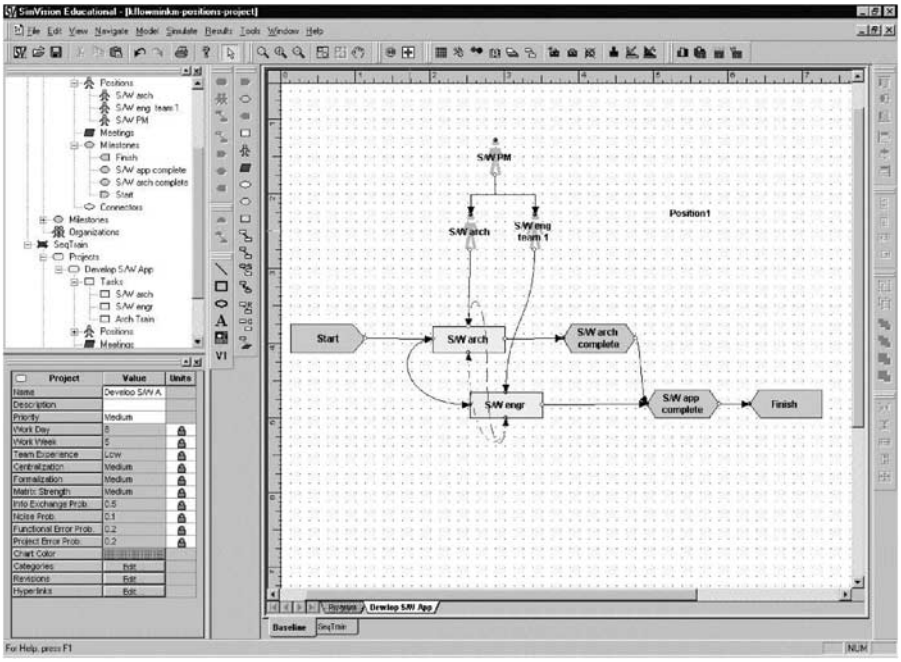

Figure 4 VDT software development model – baseline.

architecture complete, application complete, project finish) serve as markers of progress against schedule but do not involve work.

Five types of connections are delineated between organization and task elements in this representation: (1) precedence connections (e.g., between 'S/W arch' and 'S/W engr' tasks, 'S/W arch' task and 'S/W arch complete' milestone; shown in black) link tasks and milestones according to the order in which they must be accomplished; (2) communication connections (e.g., between 'S/W arch' and 'S/W engr' tasks; shown in green) link tasks associated with reciprocal interdependence, which require mutual adjustment by actors (Thompson, 1967); (3) rework connections (e.g., between 'S/W arch' and 'S/W engr' tasks; shown in red) link tasks in which exceptions or failures 'downstream' (e.g., in software engineering activities) feedback to cause rework 'upstream' (e.g., in software architecture activities); (4) assignment connections (e.g., between 'S/W eng team1' actors and 'S/W engr' task; shown in blue) indicate which organizational actors are assigned responsibility for each work task; and (5) supervision connections (e.g., between 'S/W PM' and 'S/W arch' actors) indicate the formal organizational hierarchy.

The two windows at the left of this figure reveal a tree structure of the representation (top) and numerous model parameters (e.g., priority, centralization, team experience; bottom) used to instantiate a particular work process (e.g., software development). These parameters are all set to empirically determined 'normal' values for a software development project. As noted above, these model parameters, their various settings and influences, and empirical validation of their corresponding behaviors are described in considerable detail elsewhere, so we do not repeat such description here.

The diagram in Figure 4 is representative of the kinds of models developed to date in the VDT environment. The diagram itself is clearly static. But by linking to both symbolic emulation and discrete-event simulation, the extended representation becomes dynamic. Time varying states, conditions, and results are projected day by day through the course of a modeled software development project. Because all objects, relations, and parameters in this representation are explicit, this semi-formal model of organizational behavior has relatively little ambiguity (esp. compared to textual description). As a commitment toward less-ambiguous models, the authors will be happy to send the VDT software development model described here to any interested scholar upon request. By viewing and considering in detail the many VDT parameters, settings, assumptions, and relationships made explicit through this model, little ambiguity can persist regarding what the model represents.

It is important to reiterate, the process illustrated in the figure represents a baseline case, in which only the flows of work and information are modeled. Each actor is instantiated with a specific set of skills and level of experience, which remain constant throughout the emulated period of process performance; that is, *all knowledge flows are assumed to have completed before the project begins*. We emulate the dynamics of the baseline software development workflow model below. This represents the point of departure for our present research to represent dynamic knowledge behaviors.

Baseline workflow model dynamics

As noted above, the baseline model does not include any knowledge-flow effects; that is, knowledge remains static, even though flows of work, information, and other dynamic organizational behaviors are emulated. We describe it here, and review behaviors of the corresponding VDT model, to set up a basis for comparison with the knowledge-flow models below. Table 1 summarizes some of the key emulation performance projections. Examining comparative performance represents a time-tested approach to measuring flows of knowledge (e.g., consider use of examinations in university courses). Results for the baseline workflow model are reported in Column 2. Detailed derivation and description of the various performance attributes (Column 1) can be found in Jin & Levitt (1996). We discuss key summary-level comparisons and implications here.

For instance, as shown in the table, the baseline software development work is projected to require just over 11 months to complete, and to cost $1187 K. Performance variables such as *project duration* and *cost* are quite common among contemporary simulation models. They provide summary measures for comparison of 'bottom line' project results but conceal often the factors underlying differential performance. The Schedule Growth Risk attribute (80 days) operates also at a summary level. It compares the projected duration of the project with the duration that would be achievable through Critical Path Analysis; that is, if the task were to take only the amount of time scheduled (i.e., 260 days), this variable would be zero. In the baseline case, coordination, rework and delay cause the simulated schedule for this task to grow by 80 days.

Alternatively, the rework fraction (35%) represents a lower level performance measure. It summarizes the percentage of total work associated with correcting errors and defects addressed during project performance. A complementary lower level performance measure is functional quality risk (41%). This measure summarizes the amount of additional work that would be required to correct defects that *were not addressed* during project performance. Latent defects (e.g., design errors, programming bugs) can impact adversely software product quality, and as such quality is compromised often in an attempt to reduce project rework, cost, and duration. On a behavioral level, factors such as actor skill level, task difficulty, work backlog, organization structure, and propensity to rework problems interact to influence dynamic decisions by individual actor agents about whether or not to correct errors and defects. The inherent tradeoff between cost and quality represents an important management concern. Examining emulated cost–quality results provides insights into knowledge flows that would be difficult to obtain from other research methods.

The maximum backlog variable measures organizational behavior at a lower level still. Here, it indicates the architecture actor has nearly 13 days' work in its input queue at one point in the project. Coincidentally, the software engineering team has roughly the same maximum backlog (but at a different point in the project). This dynamic measure is useful to reveal bottlenecks in a process, which can be used often to diagnose knowledge that 'clumps' in restricted areas instead of flowing as desired. Such bottlenecks can also present project managers with tough choices between cost and schedule. For instance, to ameliorate the effects of a bottleneck, additional resources (e.g., more software engineers, more skillful software architect, more team-building time for software engineers) can be added to speed the processing of work. But doing so can increase or decrease project cost. Many other performance attributes are projected through emulation. But those summarized in the table represent the key dependent variables for comparison with the knowledge-flow models below.

Modeling dynamic knowledge

In this section, the VDT modeling environment is adapted to emulate knowledge dynamics. We illustrate the use and utility of such an adaptation through agent-based modeling and emulation of knowledge flows associated with the software development work process above. We first outline the kinds of behaviors investigated to represent dynamic knowledge flows. We then instantiate and 'execute' several comparative models to illustrate such dynamic representation. We use such comparative illustration to highlight important points and contributions in terms of KM research and practice.

Dynamic knowledge behaviors

We draw from the organizational learning literature, business practice, and our own research stream to

Table 1 Emulated performance comparison

Attribute	*Baseline*	*Sequent train NL*	*Sequent train L*	*Concur train L*	*Team building*
Duration (months)	11.1	**13.6**	**12.4**	**14.6**	**11.4**
Cost ($K)	$1187	**$1212**	**$1041**	**$1125**	**$908**
Schedule growth risk (days)	80	89	**54**	**102**	**0**
Rework fraction (%)	35	33	**18**	24	**13**
Functional quality risk (%)	41	41	41	41	41
Max backlog arch (days)	13	13	**6**	**84**	5
Max backlog engr (days)	13	13	9	9	**31**

represent the dynamic knowledge. We illustrate briefly an instance from each source here. From the literature, for instance, we find that *experience* represents a highly prized and widely studied knowledge-flow phenomenon. As people and organizations gain experience performing some kind of work task, their corresponding performance of such a task generally improves over time and with repetition or practice. The learning curve (Argote, 1999) is an empirical formulation used to measure and predict such performance improvement, which can be negative (e.g., associated with forgetting, task disruption, personnel change and like organizational events; see Epple *et al*., 1991) as well as positive. Through mathematical relation of performance (e.g., measured in terms of cost, cycle time, quality) to the logarithm of *experience* (e.g., measured in terms of task repetition, practice or time), one can estimate the rate at which task performance improves through flows of knowledge associated with learning.

Studies dating back to the early days of aviation (Wright, 1936) have produced time-tested empirical relations of learning for different kinds of tasks (e.g., assembly, fabrication, procurement, negotiation), which involve knowledge flows at different rates (Ingram & Simons, 2002). We draw upon the learning curve in our VDT research to specify how knowledge flows relate to experience. Specifically, the rate of knowledge flows can be specified by a learning parameter – expressed in the form of a percentage (e.g., 80%) – to depict the dynamic performance improvement associated with each doubling of experience.

The classic expression of a learning curve is via logarithmic function: $Y = Ax^b$. Here Y refers to unit performance; x signifies cumulative experience; b is the learning parameter; and A is a constant. The learning percentage (e.g., 80%) relates directly to the learning parameter b (e.g., $b = -0.3219 = \log\ 0.80/\log\ 2$) and indicates performance in some unit of measure (e.g., cost, time, quality) will become predictably better (e.g., 20%) with each doubling of experience.

When specified as such, each sequential module developed for a software application would require predictably less time and cost as the first one did (e.g., Module 2 requires 80% as much as Module 1; Module 4 requires 80% as much as Module 2; and so forth). Factors such as increasing familiarity with the application code, understanding of the project requirements and development environment, and object reuse account in part for improvements in performance. A different learning rate may apply to each VDT actor's individual task performance, as well as performance of teams.

From business practice, as another instance, we find formal training courses are common in many organizations. Myriad knowledge workers are sent to relatively short internal or external courses to learn specific skills (e.g., use of some IT tool, maintenance procedures, employee diversity sensitivity). Such learning of specific skills represents a flow of knowledge, which can be measured by a knowledge worker's relative performance of a task before and after training. Learning specific skills *vs* general problem solving helps differentiate in part training courses from educational programs (e.g., college degrees), the latter of which are generally broader in scope and more time-consuming to complete. Such focus on specific skills also makes the effects of training courses on task performance easier to measure than the corresponding impacts of educational programs are.

Specifically, several large organizations (e.g., the U.S. Military, IBM, Motorola) have collected empirical measures of employee task performance before and after training courses are completed. We draw upon such empirical measures in our VDT research to represent the knowledge flows associated with training courses. Specifically, the magnitude of knowledge flow can be specified by adjusting the parameter *skill level* for specific actors (e.g., after completing a training course). Because different training courses can be relatively more effective in terms of knowledge flows, the skill-level parameter in VDT can be set up to change by fractional (e.g., 25, 88%) as well as unit increments. Notice such dynamic change in skill level reflects a one-time flow of knowledge, which differs from the continuous counterpart dynamics discussed above in terms of learning through experience. Clearly, there are multiple processes responsible for flows of knowledge (e.g., experience, training). Our process-level approach to representing knowledge dynamics can capture several corresponding performance effects independently.

From our research stream, as a third instance, we find groups of knowledge workers can improve their collective performance simply by working together over time. Hence a group's experience working together as a team can lead to the same kind of predictable performance improvement noted above in terms of the learning curve. However, here such experience and improvement both pertain to the group, as opposed to individuals. The VDT modeling environment has a standard, empirically validated parameter used to represent such performance effects of teams with various level of group experience. Specifically, we can represent in VDT the knowledge flows associated with such group experience by dynamic changes in the parameter *team experience* for specific work groups (e.g., a particular software development team).

Interestingly, not all experience is necessarily positive. For instance, a group of people can work together and become dysfunctional over time. In terms of knowledge flows, this could connote *negative learning*. If a group learns to work together and has a positive experience, then it would make sense to have the parameter *team experience* change for the better to reflect such experience. Alternatively, where such experience is negative, the parameter could be adjusted in the reverse direction to reflect this. A case for neutral experience arises as well. This could be addressed through no change to the parameter. Notice the similarity to the kind of 'forgetting' discussed above in reference to learning curves. This

raises the issue *negative knowledge flows*. Conceptually, we can model such flows by reversing the signs and directions of model parameters and influences. But theoretically, we need to investigate this phenomenon further. Hence we include this as a topic for future research.

Clearly, these techniques and instances represent only a beginning to representing knowledge dynamics. But they serve to illustrate the approach and suffice to instantiate several comparative dynamic knowledge models discussed below. Research to represent the dynamics of other important knowledge-flow phenomena (e.g., mentoring, education, communities of practice) continues in parallel with this adaptation of the VDT agent-based modeling environment.

Comparative dynamic knowledge models

Using these representations of dynamic knowledge behaviors, we illustrate some knowledge-flow effects through four cases derived from the baseline software development workflow model above. Each case represents a one-off representation of the baseline model; that is, in each case we modify only one parameter associated with dynamic knowledge. This provides a relatively clear basis for comparison and isolation of knowledge-flow effects. Notice such controlled manipulation of individual parameters reflects the kind of design and control associated with laboratory experimentation. Indeed, here we are able to conduct computational experiments using a validated emulation model such as VDT.

Case 1 – knowledge clumps in formal training course. The formal training approach from above is illustrated in Figure 5. Here we extend the baseline model to incorporate a specific knowledge-flow task associated with formal training. We depict the training course as a task, because it requires effort, consumes time, and is similar in most respects to other work tasks such as those corresponding to software architecture and development. In terms of knowledge distributed unevenly through the enterprise, one can view these specific knowledge work skills as 'clumped' in a training course: people must complete the course, and learn effectively, for the corresponding knowledge to flow. But a question arises as to whether the time and cost of an employee taking the course will be offset by performance improvement enabled by the corresponding knowledge flow. In this first case, we set the skill improvement parameter to zero (i.e., zero learning), which sets a lower bound for knowledge flows. This provides sharp comparison with both the baseline above and Case 2 below, in the latter of which more effective knowledge flows enable a quantum increase in skill level for the software architect.

From our theoretical framework above, formal training represents the flow of relatively explicit knowledge along the epistemological dimension, from an organization (e.g., a firm contracted to perform the training) to an individual (e.g., student) or group (e.g., class) along the ontological dimension, at the distribution phase of the life cycle. For comparison with the baseline and other cases, say the software architecture actor requires 3 months' training to learn a new tool being adopted by the enterprise for specifying architectures. The corresponding training task is represented in a fashion comparable to the other tasks (e.g., with predecessor, successor and assignment links). In this case, we assume that the training starts at project inception and that it

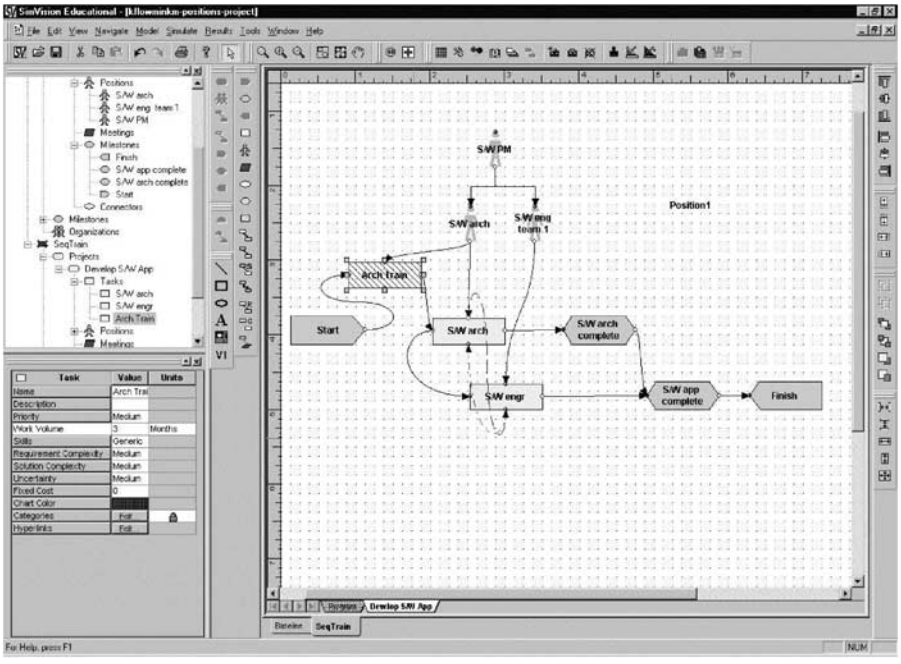

Figure 5 VDT software development model – with sequential architecture training.

must be completed before the software architecture task can begin. Predictably, the schedule for the software development project is extended by about 3 months. Emulated performance results for this case are presented in Column 3 of the table (i.e., labeled 'Sequent train NL') above for comparison with the baseline model.

As would be expected for addition of a new sequential task, both the project duration (i.e., now 13.6 months) and cost (i.e., now \$1212 K) reflect appreciable change; these are highlighted in bold print in the table. Some of the other values are slightly different, due principally to random variations in the emulation runs and the addition of the training task. Despite modeling dynamically the additional time and cost associated with this knowledge-flow process, however, the case does not reflect any *benefit* in terms of new knowledge acquired by the software architecture actor; that is, this scenario does not show improvement in the actor's skill level. Clearly, if this situation could be anticipated, there would be little reason to invest in the training course.

Case 2 – knowledge flows through formal training course. The fourth column in Table 1 (i.e., labeled 'Sequent train L') includes the knowledge-flow benefit: after completing the training course, the software architect actor has a quantum increase in skill level (e.g., from 'medium' to 'high') and performs better on the architecture task. In other words, here we change the skill-level parameter to reflect knowledge flows associated with the training course. In other words, knowledge that was 'clumped' in the training course for Case 1 above has flowed to the software architect actor here in Case 2. Notice the knowledge-flow (i.e., learning) effect brings the schedule growth down to 54 days. Both project duration (i.e., 12.4 months) and cost (i.e., \$1041 K) reflect this learning also. Notice too the level of rework is down considerably (i.e., to 18%), as is the architecture actor's backlog (i.e., to 6 days). These effects derive from the improved knowledge flows and corresponding work performance improvement of the architect.

Alternatively, this training course has no direct effect on the software engineering team or performance of the other project task; that is, the knowledge flow is restricted to the individual architect actor. Hence the overall simulation results do not show dramatic improvement. Indeed, even with the knowledge-flow effect, the project takes longer to complete with the training course than in the baseline case summarized above. This represents another good point of comparison.

Case 3 – knowledge flows through concurrent formal training and project work. Case 3 continues with the architecture training example from above. Figure 6 shows the same situation depicted in Case 2 (i.e., actor takes the training course; actor's skills improve), except to 'save time,' the software architect actor is expected to accomplish the training concurrently (e.g., via an online course accomplished in the office) with its assigned project work task. This corresponds to a case common in practice, where a project leader figures such approach will reduce project time through concurrent training and project work. From the screenshot presented in the figure, this scenario reveals just one difference with that in Figure 5: the architecture work task no longer has to wait for the architecture training course to be completed. Instead, architecture training (i.e., knowledge flows)

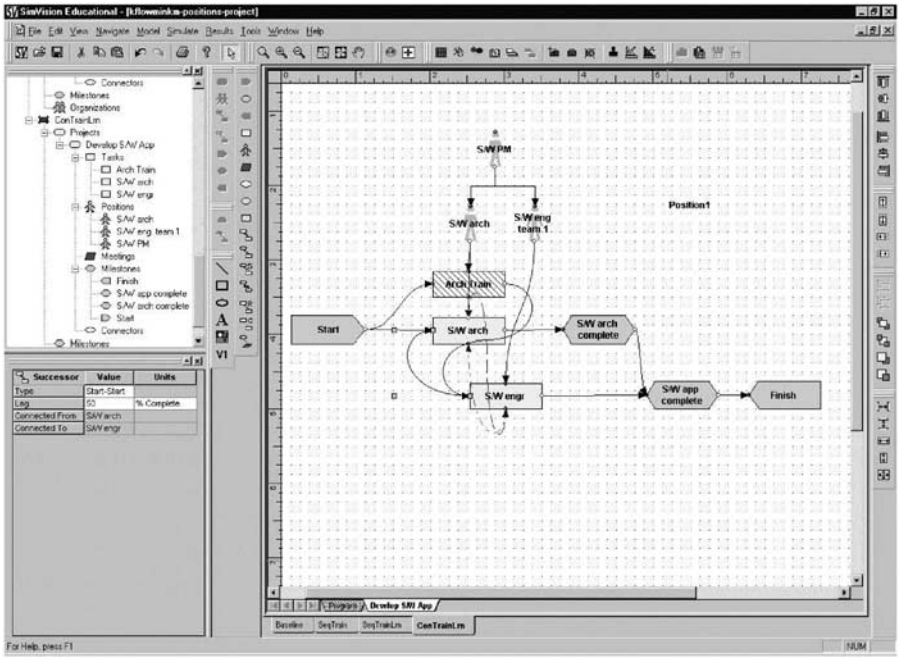

Figure 6 VDT software development model – with concurrent architecture training.

and architecture development (i.e., workflow) are now performed concurrently. Other than this, the models are identical.

The simulated performance is summarized in Column 5 of the table (i.e., labeled 'Concur train L') above. Notice the project duration increases to 14.6 months, and project cost increases to $1125 K. Contrary to the project leader's plan, requiring the software architect actor to learn the new architecture specification system while performing the architecture development task itself would require more time and money than sending the person to the course for 3 months would. This kind of result is not apparent from the static representation. Nor would concurrent work and training such as this necessarily always extend project cost and schedule. Rather, numerous factors – such as length of the training course, learning efficacy of the architect actor, difficulty of the architecture task, concurrency of the software architecture and engineering tasks, level of project staffing, organization structure, and others – interact in a complex manner that would be difficult to evaluate without a computational model. Cause–effect interactions such as these can defy managerial intuition often.

For instance, notice the backlog for the software architect actor has increased to 84 days. Consistent with practical experience, when a knowledge worker is required to accomplish two tasks in concurrence – as opposed to serially – a boundedly rational individual (e.g., limited cognitive capability) with constrained resources (e.g., a 40-h workweek) may experience more difficulties *with both tasks* and expend more time and effort overall. A manager in practice could use the VDT model to analyze computationally the sensitivity of this result by trying other managerial interventions such as authorizing overtime (e.g., 60-h workweeks) for the architect, reducing concurrency between the software architecture and engineering tasks, hiring or assigning a more experienced software architect to the task (i.e., to obviate the need for training), and others.

Further, we can stop the VDT model emulation at arbitrary points and examine various factors of interest to gain additional insight into the dynamics of knowledge flows. For instance, when the architecture training course first begins, one can analyze the situation: the architect actor's backlog is low; this actor is able to keep up with its communications and work tasks; this actor is completing the training coursework on schedule; and the team of engineering actors is not impacted by the architect's training course. Hence, the project is progressing according to plan. We restart the emulation and stop it again in midcourse to see how the situation has changed: the architect actor gets increasingly behind in terms of both assignments for the training course and work activities for the architecture development tasks; errors and miscommunications begin to accumulate, requiring increasing levels of rework and retransmission; the engineering actors must wait longer for outputs from the architecture task; and the architecture actor does not increase its skill level as rapidly as in the case above. Hence the project is falling well behind schedule now. These represent fine-grained, dynamic, knowledge-flow effects that would be difficult to observe – particularly in a longitudinal manner – without being able to stop, analyze and replay the computational model.

Case 4 – knowledge flows through formal training and team building. Figure 7 displays our final case, in which the

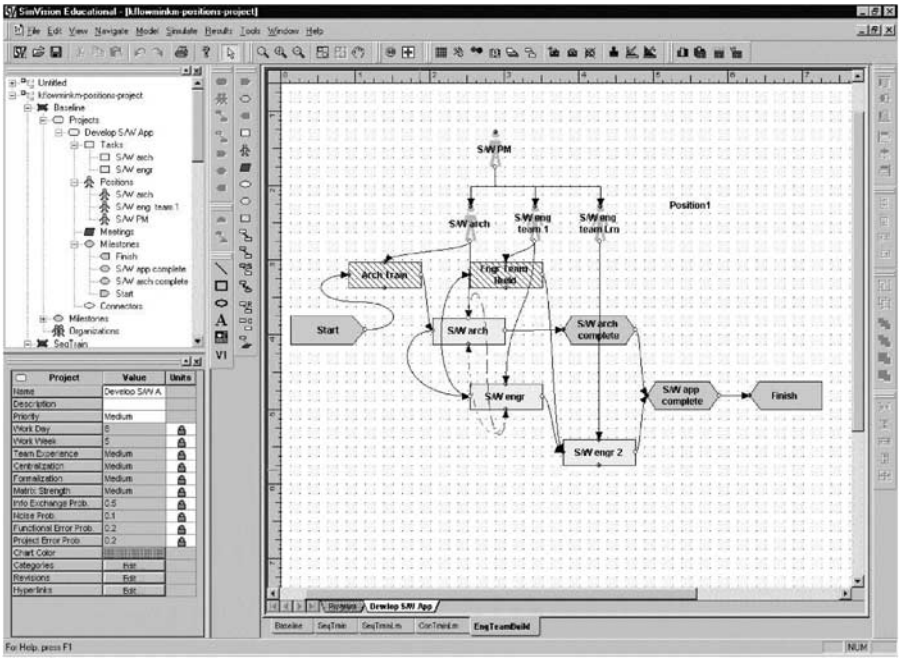

Figure 7 VDT software development model – with engineer team building.

equivalent of 1 month's time each (i.e., 10 person-months in total) is allocated for members of the software engineering team to meet, share experiences, and engage in the kinds of non-project activities associated often with a community of practice. This instantiates the socialization process articulated by Nonaka. As with the architecture training course discussed in the cases above, this team building activity pertains to the flow of knowledge as opposed to the flow of work. Indeed, many managers (and organization scholars) would consider such activity to be non-productive and such time to be wasted. Yet other managers (and scholars) would anticipate performance benefits in terms of team learning. As above, it is unclear whether the investment in team building will be offset by productivity gains in team performance. We include here also the effects of individual-level knowledge flows through learning curves associated with experience; that is, we model both group-level knowledge flows through team learning and individual-level flows through experience.

Specifically, we separate the software engineering task into two phases – one can think of these as separate *releases*, a common practice in software development – and we represent the software engineering team differently for each release. The first team of actors (i.e., labeled 'S/W eng team1') is responsible for Release 1 of the software and is specified as before: the software development group has low team experience; individual actors in the group have medium-level software engineering skills. This same team of actors – but at a later point in time, and benefiting from experience working individually and together (i.e., labeled 'S/W eng team Lrn') – is responsible for Release 2 of the software and is specified to reflect the dynamics of (positive) knowledge flows: the software development group develops higher team experience; individual actors in the group develop high-level software engineering skills. The other elements delineated in Figure 7 remain the same as in the sequential training case above (i.e., Case 2).

Notice the simulated results in Column 6 of Table 1 (i.e., labeled 'Team building'). Despite introducing some slack into the software engineers' work schedule, the overall project duration contracts to 11.4 months. This is roughly equivalent to the original schedule that included no knowledge-flow processes (e.g., no architecture training, no team building). Further, because the knowledge and performance levels of both the software architect actor and software engineering team have increased, the project cost *is less than the baseline* (i.e., $908 K). One key is the rework statistic, which has dropped to 13%. Notice also the schedule growth of zero. This indicates that the project has now converged onto the critical path: another improved result over the original project baseline. Alternatively, one can see the effect of the engineers' investment in team building through its increased backlog (i.e., 31 days). This quantifies the intuitive notion of mounting project work that accumulates as team members are engaged in non-project (i.e., team building) knowledge-flow activities.

In this particular case, the positive knowledge-flow effects of team building more than compensate for the negative effects of 'non-productive' time invested in such team building. This assumes that the team composition remains relatively stable through both software releases and that neither the software development tools nor major requirements of the software product change abruptly during this time. It also assumes that the team as a whole, and each software engineer as an individual, effects a quantum increase in experience. Personnel turnover would impact negatively such learning, as would major changes in the work environment or requirements. Clearly where the group is not effective at learning to work as a team, or where individuals fail to learn as predicted, the knowledge-flow benefits would be less (or possibly even negative).

It should also be clear, many of the factors examined through these cases are under management control and depend upon workers' collective goals as well as organization designs and technologies. We now have a method and tool to examine explicitly the knowledge-flow effects of different organization designs, tools, teams and management approaches. The researcher gains new insight into how to separate performance effects of knowledge flows and workflows. The manager gains new insight into performance tradeoffs between investing in workflows and investing in knowledge flows.

Additionally, through agent-based modeling and emulation, we improve our understanding of how knowledge flows differ from flows of work and information. We also understand better how various knowledge-flow processes (e.g., learning by experience, formal training, group interaction) affect work performance in different ways and with different impacts. By stopping the model emulation, and analyzing relatively fine-grained variables (e.g., work backlog, requests for information, rework), we are able to develop new knowledge of *how* flows of knowledge affect flows of work and information. By using computational models that reflect well-established organization theory as input and that benefit from extensive empirical validation of output, we also develop confidence that emulation results such as these – along with the research and managerial capabilities they enable – can generalize well to operational organizations in practice. This represents a contribution to new knowledge in the KM field.

Conclusion

Knowledge is distributed unevenly through most enterprises. Hence flows of knowledge (e.g., across time, people, locations, organizations) are critical to organizational efficacy and performance under a knowledge-based view of the firm. However, supported principally by narrative textual theory in the emerging knowledge management (KM) field, the researcher has difficulty describing how different kinds of knowledge will flow

through various parts of an organization. This causes difficulty also for predicting the effects of alternate approaches to dispersing knowledge that 'clumps' in various areas. This problem is also manifest for the KM professional, who lacks clear theory or tools to anticipate how any particular information technology or other managerial intervention may enhance or impede specific knowledge flows in the enterprise. In this expository article, we build upon a steady stream of research in computational organization theory to develop agent-based models of knowledge dynamics. This work draws from emerging theory for multidimensional representation of the knowledge-flow phenomenon, which enables the dynamics of enterprise knowledge flows to be formalized and emulated through computational models. This approach provides the means for knowledge-flow processes to be visualized and analyzed in new ways. Computational experimentation enables the performance of many alternate process designs and technological interventions to be compared through examination of dynamic models, before committing to a specific approach in practice. We illustrate this research method and modeling environment through semi-formal representation and agent-based emulation of several knowledge-flow processes from the domain of software development. We also outline key directions for the new kinds of KM research and practice elucidated by this work.

The VDT agent-based modeling environment continues its development and refinement to incorporate better the kinds of dynamic knowledge-flow behaviors described above. When this research associated with knowledge flows began, for instance, the VDT environment had to be 'tricked' into emulating the effects of learning, training, group interaction, and other related phenomena. The next major version of VDT, which is under development at the time of this writing, has been specified to incorporate such knowledge-flow behaviors through simple parameterization, and we continue investigating the effects of other knowledge-flow phenomena such as mentoring, education, and communities of practice. We plan to investigate also the phenomenon *forgetting* and the class of behaviors associated with *negative knowledge flows*.

However, our key point in describing VDT and several of its agent-based models above is not to showcase VDT or to suggest that the tool is necessary for understanding knowledge-flow dynamics. Rather, we use VDT to help us understand such dynamics *better*. We can isolate specific learning effects and emulate the dynamic knowledge behaviors of myriad different organizations – many of which may not yet exist in concept or practice – in a manner analogous to controlled laboratory experimentation. Through such isolation, emulation and computational experimentation, we gain the ability to examine semi-formal models of organizations, to stop the action of a process for examination, to replay the same process many times, and to visualize the dynamics of knowledge flows in a new way. It is through the *use* of a tool such as VDT that we are learning more about the dynamics of knowledge flows. Such *learning*, not the tool, represents our primary contribution through this article.

References

ALAVI M and LEIDNER DE (2001) *Review*: knowledge management and knowledge management systems: conceptual foundations and research issues. *MIS Quarterly* **25(1),** 107–136.

ARGOTE L (1999) *Organizational Learning: Creating, Retaining and Transferring Knowledge*. Kluwer, Boston.

AUGIER M, SHARIQ SZ and VENDELO MT (2001) Understanding context: its emergence, transformation and role in tacit knowledge sharing. *Journal of Knowledge Management* **5(2),** 125–136.

CARLEY KM and PRIETULA MJ (Eds) (1994) *Computational Organization Theory*. Lawrence Erlbaum Associates, Hillsdale, NJ.

CARROLL T and BURTON RM (2000) Organizations and complexity: searching for the edge of chaos. *Computational & Mathematical Organization Theory* **6(4),** 319–337.

CHENG CHF and LEVITT RE (2001) Contextually changing behavior in medical organizations. *Proceedings of the 2001 Annual Symposium of the American Medical Informatics Association,* Washington, DC, November 3–7, 2001.

CHRISTIANSEN TR (1993) Modeling efficiency and effectiveness of coordination in engineering design. Teams doctoral dissertation, Department of Civil and Environmental Engineering, Stanford University.

COHEN GP (1992) The virtual design team: an object-oriented model of information sharing in project teams. Doctoral dissertation, Department of Civil Engineering, Stanford University.

COLE RE (1998) Introduction. *California Management Review* **45(3),** 15–21.

DAVENPORT TH (1993) *Process Innovation: Re-engineering Work through Information Technology*. Harvard University Press, Boston, MA.

DAVENPORT TH, DE LONG DW and BEERS MC (1998) Successful knowledge management projects. *Sloan Management Review* **39(2),** 43–57.

DAVENPORT TH and PRUSAK L (1998) *Working Knowledge: How Organizations Manage What They Know*. Harvard Business School Press, Boston, MA.

DESPRES C and CHAUVEL D (1999) Mastering information management: part six – knowledge management. *Financial Times* 4–6.

DIXON NM (2000) *Common Knowledge*. Harvard Business School Press, Boston, MA.

EPPLE D, ARGOTE L and DEVADAS R (1991) Organizational learning curves: a method for investigating intra-plant transfer of knowledge acquired through learning by doing. *Organization Science* **2(1),** 58–70.

GALBRAITH JR (1977) *Organization Design*. Addison-Wesley, Reading, MA.

GARTNER GROUP (1998) Knowledge management scenario. *Conference presentation* Stamford, CN, presentation label SYM8KnowMan-1098Kharris.

GRANT RM (1996) Toward a knowledge-based theory of the firm. *Strategic Management Journal* **17,** 109–122.

INGRAM P and SIMONS T (2002) The transfer of experience in groups of organizations: implications for performance and competition. *Management Science* **48(12),** 1517–1533.

JIN Y and LEVITT RE (1996) The virtual design team: a computational model of project organizations. *Computational and Mathematical Organization Theory* **2(3),** 171–195.

KING WR and KO DG (2001) Evaluating knowledge management and the learning organization: an information/knowledge value chain approach. *Communications of the Association for Information Systems* **5(14),** 1–27.

LAW AM and KELTON D (1991) *Simulation Modeling and Analysis*, 2nd Edn. McGraw-Hill, New York, NY.

LEAVITT HJ (1965) Applying organizational change in industry: structural, technological and humanistic approaches. In *Handbook of Organizations* (MARCH J, Ed), Rand McNally, Chicago, IL.

LEVITT RE, THOMSEN J, CHRISTIANSEN TR, JUNZ JC, JIN Y and NASS C (1999) Simulating project work processes and organizations: toward a mocro-contingency theory of organizational design. *Management Science* **45(11),** 1479–1495.

KUNZ JC, LEVITT RE and JIN Y (1998) The virtual design team: a computational simulation model of project organizations. *Communications of the Association for Computing Machinery* **41(11),** 84–92.

MARCH JG and SIMON HA (1958) *Organizations*. John Wiley, New York.

NISSEN ME (1999) Knowledge-based knowledge management in the re-engineering domain. *Decision Support Systems. Special Issue on Knowledge Management* **27,** 47–65.

NISSEN ME, KAMEL MN and SENGUPTA KC (2000) Integrated analysis and design of knowledge systems and processes. *Information Resources Management Journal* **13(1),** 24–43.

NISSEN ME (2002) An extended model of knowledge-flow dynamics. *Communications of the Association for Information Systems* **8,** 251–266.

NOGUEIRA JC (2000) A formal model for risk assessment in software projects. Doctoral dissertation, Department of Computer Science, Naval Postgraduate School.

NOLAN NORTON INSTITUTE (1998) Putting the knowing organization to value. White paper (August 1998), cited in Alavi and Leidner (2001).

NONAKA I (1994) A dynamic theory of organizational knowledge creation. *Organization Science* **5(1),** 14–37.

NONAKA I, TAKEUCHI H and UMEMOTO K (1996) A theory of organizational knowledge creation. *International Journal of Technology Management, Special Issue on Unlearning and Learning for Technological Innovation* **11(7/8),** 833–845.

O'LEARY DE (2001) How knowledge reuse informs effective system design and implementation. *IEEE Intelligent Systems* (January/February), 44–49.

RYLE G (1958) *The Concept of Mind*. Hutchinson, London.

SCHULTZE U and BOLAND RJ (2000) Knowledge management technology and the reproduction of knowledge work practices. *Strategic Information Systems* **9,** 193–212.

SPENDER JC (1996) Making knowledge the basis of a dynamic theory of the firm. *Strategic Management Journal* **17,** 45–62.

SWAP W, LEONARD D, SHIELDS M and ABRAMS L (2001) Using mentoring and storytelling to transfer knowledge in the workplace. *Journal of Management Information Systems* **18(1),** 95–114.

SZULANSKI G (2000) The process of knowledge transfer: a diachronic analysis of stickiness. *Organizational Behavior and Human Decision Processes* **82(1),** 9–27.

TEECE DJ (1998) Research directions for knowledge management. *California Management Review* **40(3),** 289–292.

THOMSEN J (1998) The virtual team alliance (VTA): modeling the effects of goal incongruency in semi-routine, fast-paced project organizations. Doctoral dissertation, Department of Civil and Environmental Engineering, Stanford University.

THOMPSON JD (1967) *Organizations in Action: Social Science Bases in Administrative Theory*. McGraw-Hill, New York.

VDT (2004) The Virtual Design Team Research Group website; URL: http://www.stanford.edu/group/CIFE/VDT/.

VENZIN M, VON KROUGH G and ROOS J (1998) Future research into knowledge management. In *Knowing in Firms: Understanding, Managing and Measuring Knowledge* (VON KROUGH G, ROOS J, and KLEINE D, Eds), pp 26–66. Sage, London.

VON HIPPEL E (1994) 'Sticky information' and the locus of problem solving: implications for innovation. *Management Science* **40(4),** 429–439.

WONG SS and BURTON RM (2000) Virtual teams: what are their characteristics, and impact on team performance? *Computational & Mathematical Organization Theory* **6(4),** 339–360.

WRIGHT TP (1936) Factors affecting the cost of airplanes. *Journal of Aeronautical Sciences* **3(4),** 122–128.

ZACK M (1998) What knowledge-problems can information technology help to solve. In *Proceedings Americas Conference on Information Systems* (HOADLEY E and BENBASAT I, Eds), pp 644–646, Baltimore, MD.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.