Association for Information Systems
AIS Electronic Library (AISeL)

ICIS 2009 Proceedings

International Conference on Information Systems (ICIS)

2009

Learning and Forgetting Curves in Software Development: Does Type of Knowledge Matter?

Keumseok Kang
Purdue University, kangk@purdue.edu

Jungpil Hahn
Carnegie Mellon University, jungpil@cmu.edu

LEARNING AND FORGETTING CURVES IN SOFTWARE DEVELOPMENT: DOES TYPE OF KNOWLEDGE MATTER?

Completed Research Paper

Keumseok Kang
Krannert School of Management
Purdue University
West Lafayette, IN 47907
kangk@purdue.edu

Jungpil Hahn
Tepper School of Business
Carnegie Mellon University
Pittsburgh, PA 15213
jungpil@cmu.edu

Abstract

What type of knowledge, among domain, technology, and methodology knowledge, is most influential to the performance of software development? We answer to this question by empirically investigating the learning and forgetting curves in software development using an extensive archival data set of software development projects in an IT service company. We find that prior experiences with the same methodology or technology have a stronger impact on software project performance than those in the same application domain. Furthermore, our results show that methodology knowledge is more easily forgotten than domain or technology knowledge. Our findings provide managerial implications not only to the development of knowledge and skills, but also to other organizational issues in software development such as project team staffing and career development.

Keywords: Learning curves, forgetting curves, software development, procedural knowledge, declarative knowledge, domain knowledge, technology knowledge, methodology knowledge

Introduction

Developing organizational software (or software development) is a very complex activity because it must deal with people, organizations, technologies, and business processes (Brooks 1987). For this reason, software project management has been considered as one of the most complex problem domains and has been extensively studied for decades. However, recent statistics suggest that the software industry still suffers from high failure rates due to poor project management. For example, a recent report by the Standish Group notes that in 2004, only 18% of software projects were considered successful, whereas 53% were challenged and 18% were considered failures, with average cost and schedule overruns of 56% and 84%, respectively.

To better provide managerial guidance to software development practices, recent research has begun to adopt a knowledge-based perspective of software development by applying theories from the cognitive and organizational sciences to investigate various aspects of software development. For example, cognitive theories of knowledge transfer have been used to investigate the transition from traditional data- and process-oriented development paradigms to object-oriented development (Armstrong and Hardgrave 2007); organizational theories of coordination have been used to investigate the role of transactive memory in software development performance (Faraj and Sproull 2000). In essence, this new stream of research focuses on knowledge to provide theoretical explanations of and implications to software practices. The main focus of the knowledge perspective is to understand how knowledge is acquired, shared, and coordinated; and, how this may impact software development practices at the individual, project and organizational levels.

If knowledge plays a key role in software development, understanding how knowledge processes impact software development performance becomes very critical to software organizations. Recent research has documented the existence of the learning curve in software development and maintenance (Boh, Slaughter, & Espinosa 2007; Huckman, Staats, & Upton 2009; Langer, Slaughter, & Mukhopadhyay 2008; Narayanan, Balasubramanian, & Swaminathan 2009). In other words, these studies have shown that software teams (or individual developers) become more productive as they accumulate experiences on *similar* software projects. Although the documentation of learning curve effects by these initial studies has important implications for software development practice, these studies have several limitations that deserve further research so as to provide more substantive managerial guidance to software practitioners.

The core mechanism of learning curve theory is that performance gains materialize with repeated application of the same unit task. However, the context of software development is more complex and frequently presents a number of situations where the assumptions of the learning curve may no longer apply. First, in software development projects, the activities and tasks may be related and similar but are rarely the same across projects. Fortunately, prior studies have shown that learning curve effects exist for repeated *similar* or *related* project experiences (Schilling, Vidal, Ployhart, & Marangoni 2003; Boh et al. 2007; and Narayanan et al. 2009), even if the projects are not exactly the same.

Second, software projects are typically quite complex and require a variety of multidisciplinary knowledge and skills. For instance, in order to develop an online banking (e-banking) system for a retail bank, the software team must understand the target bank's business processes (i.e., knowledge of the domain), the platform technologies (e.g., programming languages, database management systems, network architectures, security, etc.) that will be used to implement the system (i.e., knowledge of technologies), and also the processes to design and build the target software (i.e., knowledge of software development methodologies). Each individual software project (in a particular domain, using a particular set of technologies and development methodologies) requires a different set of knowledge and skills that fit the context at hand. Whether learning curve effects exist for the different types of knowledge and skills is an important research question with substantial practical implications for project staffing and professional development for software practitioners. For example, if (hypothetically) learning curve effects were to exist for domain knowledge but not for technology knowledge or methodology knowledge, then software organizations should nurture specialization in business domains but not in technologies or software development methodologies. The prior literature unfortunately has not scrutinized the learning effects for different types of software development knowledge and skills.

Third, software developers may accumulate experience by doing similar or related projects over time (which would lead to performance gains if learning curve effects exist). However, this cumulative experience does not persist but depreciates over time. Moreover, due to the huge variety of software projects, typically developers are engaged in

unrelated projects and exposed to various software development knowledge and skills, which may facilitate this depreciation. In other words, current practices for project staffing have much room and potential for interference and subsequent knowledge degradation (i.e., forgetting), which may ultimately hinder organizational productivity.

In this paper, we apply the cognitive theory of learning and forgetting to investigate whether different types of software development knowledge and skills (i.e., domain vs. technology vs. methodology knowledge) exhibit differential learning curves as well as differential forgetting curves. Since learning and forgetting effects are evidenced as performance improvements and degradation, understanding learning and forgetting effects has direct implications for enhancing software productivity. In addition to these direct benefits to software productivity, our study has implications for other important practical considerations such as short-term project staffing and long-term career development paths for software professionals.

The remainder of the paper is organized as follows. In the next section, we present our theoretical background and develop our research hypotheses. Next we outline the research methodology, which includes details of the dataset, measures, and analytical approach. This is followed by our analyses and results. Finally, we conclude with a discussion of the results, limitations, managerial implications, and suggestions for future research.

Theory and Hypotheses

Software Development Knowledge

As discussed earlier, developing software requires a variety of knowledge and skills. A number of authors have classified the different types of knowledge and skills. For example, Lee et al. (1995) categorized critical knowledge and skills for IS professionals into technical specialty, technology management, business functional knowledge, and interpersonal and management skills. Abraham et al. (2006), Goles et al. (2008), and Simon et al. (2007) classified the IT workforce skills into project management, business domain, technical skills, and general management skills. Chan et al. (2008) classified the knowledge and skills required in IS development into application domain and development methods skills. Lastly, Langer et al. (2008) defined hard skills as technological knowledge, domain knowledge, and experience. Although some inconsistencies exist in the terminology used (see Table 1), the different knowledge and skills required to develop software can be categorized into three types – 1) knowledge of the business domain (i.e., domain knowledge), 2) knowledge of the technology used to implement the software (i.e., technology knowledge), and 3) knowledge related to the process of conducting a software project (i.e., methodology knowledge).

Table 1. Taxonomies of Software Development Knowledge

Reference	Domain Knowledge	Technology Knowledge	Methodology Knowledge
Lee et al. (1995)	Business functional knowledge	Technical specialty and technology management	Interpersonal and management skills
Abraham et al. (2006), Simon et al. (2007), Goles et al. (2008)	Business domain	Technical skills	Project management
Chan et al. (2008)	Application domain	Development methods and skills	
Langer et al. (2008)	Domain knowledge	Technological knowledge	Experience

First, *domain knowledge* refers to the knowledge about the target application domain of the software to be built and the context in which that software will be used. For example, developing a production planning system for a manufacturing company requires developers to understand how to create and manage production plans and how the production plan fits into the overall production process. Similarly, developing an application for the financial services industry requires knowledge of financial instruments, pricing models, etc. As such, domain knowledge plays a critical role in software development – without domain knowledge, one would not know what to build into the software system.

Ultimately, software systems must be implemented with technology. *Technology knowledge* relates to knowledge on what various technologies are, how they work, and how to implement them. As with domain knowledge, each software project uses a variety of technologies in terms of technical architectures (e.g., mainframe, client-server,

Web-based, mobile etc.), programming languages (e.g., procedural languages such as COBOL, Pascal, and C, object-oriented languages such as Java, C++, and C#), database technologies (e.g., file systems such as ISAM, relational and object-relational databases, object-oriented databases etc.), among others.

Finally, software development is a complex endeavor that necessitates effective management of the development process. Consequently, knowledge of how to conduct and manage the various development activities, which we refer to as *methodology knowledge*, is also essential for software projects. Like technologies, there are many different methodologies currently used in practice and each methodology requires a different set of knowledge and skills. For instance, traditional waterfall-based methodologies such as Information Engineering (Martin, 1989) are quite different from iteration based methodologies like the Rational Unified Process (RUP) or the more recently proposed agile methodologies.

Learning Curves

Learning, which is the accumulation of knowledge and skills, typically occurs through repeated experiences (or trials). The theory of learning curves aptly portrays the relationship between amount of experience and performance gains. The existence of learning curves means that cumulative experiences lead to increased performance. Therefore, learning curves can be equated with learning from experience (i.e., learning by doing). Ever since the first documentation of organizational learning curves in aircraft production (Wright 1936), learning curves have been found in a wide variety of industries (see Yelle (1979), Dutton et al. (1984), and Argote and Epple (1990) for reviews).

Despite the aforementioned characteristics of the software development context (e.g., similar but different tasks, various knowledge and skills, potential for interference of unrelated tasks between trials, etc.) that make it unfavorable for efficient learning, several recent studies have documented learning curve effects in software development. Boh et al. (2007) were the first to document learning curve effects in software maintenance; Huckman et al. (2009) further showed that role experience amplified the learning effect; Narayanan et al. (2009) reported individual learning effects in software maintenance environment. While these studies have successfully documented the existence of learning curve effects in software development, they offer limited managerial implications as knowledge and experiences were classified at the project level (e.g., related projects vs. unrelated projects) or were not classified at all (i.e., all projects are same). Given that software development requires a diverse set of knowledge and skills – domain knowledge, technology knowledge and methodology knowledge, and in practice, project staffing typically considers prior experiences in at the component knowledge level (rather than at the project level), it is important to ascertain whether learning curve effects exist independently for different components of software development knowledge and skills. We first hypothesize:

H1a: *A software project team's performance on a current project is positively affected by the amount of prior experience in working in the same domain.*

H1b: *A software project team's performance on a current project is positively affected by the amount of prior experience in working with the same technology.*

H1c: *A software project team's performance on a current project is positively affected by the amount of prior experience in working with the same methodology.*

Differential Learning Curves in Software Development: Procedural vs. Declarative Knowledge

Since the underlying mechanism of the learning curve is learning, it is possible to observe differential learning curves for different learning rates. Several studies have explained why different learning curves exist even in the same context. Pisano et al. (2001) argued that learning curves may vary by characteristics of organization whereas Reagans et al. (2005) showed that a firm's learning rate is affected by individual workers' task proficiency, the manager's ability to leverage workers' knowledge, and the organization's capacity to coordinate work activities. Schilling et al. (2003) and Wiersma (2007) found that related experiences produce faster learning rates (or steeper learning curves) than unrelated experience that are overly generalized and the same experience that are overly specialized for the task at hand. Wiersma (2007) further argued that learning effects can be enhanced with slack in resources. As can be seen from this brief review of the differential learning curve literature, past research has focused primarily on factors relating to *who* learns (i.e., the individual workers (or groups) and the relationship between individuals), and *how* they learn (i.e., the structure of work practices). Interestingly, little attention has

been devoted to the actual contents of *what* is learned. Given that software development involves a variety of knowledge and skills (i.e., knowledge of domain, technology and methodology), we focus our attention on the characteristics of knowledge itself, which may affect learning rates.

Some knowledge and skills are learned faster by doing. For example, learning the capitals of nations is quite different from learning to drive a car. Driving a car requires real-world practice and hands-on experiences whereas memorizing the capitals does not. Knowledge can be categorized as either *procedural or declarative* (Anderson 1982; Gagne, Yekovich, & Yekovich 1985; McCormick 1997). Procedural knowledge, frequently referred to as "know how", is the knowledge about how to perform a task. Performance capabilities such as the ability to write, read, or solve the algebra problems fall into this category. Procedural knowledge is usually acquired by (repeatedly) exercising the task. On the other hand, declarative knowledge, also called "know what", is the knowledge about that something is the case. Declarative knowledge is, by its very nature, expressed in declarative sentences or indicative propositions and is exemplified by organized collections of facts and concepts (Anderson 1983; Jones and Idol 1990).

Although individuals can learn both procedural and declarative knowledge from experience, the effectiveness of experiential learning (i.e., learning by doing) is not the same due to the intrinsic differences between the two knowledge types. Procedural knowledge is about how to perform a given task, and so learning from experience is typically the more effective way to acquire this type of knowledge. On the other hand, declarative knowledge is the knowledge about what something is. Therefore, logical reasoning or in-class instruction can be more appropriate for learning declarative knowledge than learning by doing (Gagne et al. 1985). In summary, although both procedural and declarative knowledge can be acquired by learning by doing; procedural knowledge is expected to have a steeper learning curve than declarative knowledge.

Given the differential experiential learning rates for procedural and declarative knowledge, the question becomes which among domain, technology and methodology knowledge, can be categorized as procedural or declarative. Methodology knowledge consists of the processes involved in building a software system and knowledge about how to conduct each of the processes. Domain knowledge, on the other hand, consists of business entities, functions, and processes and the relationships among them. Consequently, since domain knowledge represents *what* to develop (i.e., a collection of fact and concepts about the business domain) and methodology knowledge represents *how* to develop the software, it follows that methodology knowledge can be conceptualized as being closest to procedural knowledge and domain knowledge as closest to declarative knowledge. Bassellier and Benbasat (2004) used a similar conceptualization wherein business domain knowledge was characterized as declarative knowledge and know-how and skills as procedural knowledge in IS development.

Classifying technology knowledge is a little trickier as technology knowledge contains characteristics of both procedural and declarative knowledge. Vincenti (1984) and Herschbach (1995) classified software development knowledge into three types – descriptive, prescriptive, and tacit[1] knowledge – and argued that technology has characteristics of all three knowledge types. Here, descriptive knowledge is synonymous to declarative knowledge, whereas prescriptive knowledge is synonymous to procedural knowledge. Lee et al. (1995) defined two types of technical knowledge required for IS professionals – technical specialties and technical management. Technical specialties are concerned with the technologies themselves, whereas technical management refers to the knowledge and skills concerned with where and how to deploy technologies effectively to meet project objectives. It seems that technology cannot be simply classified as belonging to either declarative or procedural knowledge; rather technology knowledge contains characteristics of both types.

That being said, technologies contain more declarative aspects than methodologies, which can be evidenced by the fact that specifications for technology (i.e., what is it and how it needs to used and when, etc) are more extensive, more complex, and more explicitly documented than those for methodologies. Technology knowledge, however, contains more procedural knowledge than domain knowledge. For example, although programming languages has explicit syntax and rules (i.e., declarative aspects of programming languages), real-world experience is typically required to effectively and efficiently use them in practice (i.e., procedural aspects of programming languages). It

[1] The term "tacit" as used in Vincenti (1984) and Herschbach (1995) actually refers to procedural knowledge or "know how". This should not be confused with the term "tacit" used frequently in the knowledge management literature to reflect the difficulty in externalization and codification of knowledge (Polanyi 1967; Nonaka 1991).

can be argued that technology knowledge resides in between domain knowledge and methodology knowledge along the declarative – procedural knowledge continuum.[2]

Given that procedural knowledge is acquired through experience more efficiently than declarative knowledge, and that methodology knowledge can be characterized as procedural knowledge, domain knowledge as declarative knowledge, and technology knowledge as a hybrid form composed of both procedural and declarative knowledge, we hypothesize that the learning curve effects related to prior experiences with the same methodology will be stronger than those for experiences with the same technology, which will in turn be stronger for those for experiences with the same domain. In other words, we expect learning curves to be steepest for methodology knowledge, then technology knowledge and finally domain knowledge. More formally, we hypothesize:

H2a: *The positive impact of the amount of prior experience working with the same methodology is stronger than the impact of the amount of prior experience working in the same domain.*

H2b: *The positive impact of the amount of prior experience working with the same technology is stronger than the impact of the amount of prior experience working in the same domain.*

H2c: *The positive impact of the amount of prior experience working with the same methodology is stronger than the impact of the amount of prior experience working with the same technology.*

Forgetting Curves

Although learning reflects an accumulation of knowledge and skills, knowledge depreciates over time. Forgetting refers to such depreciation of knowledge and typically occurs when the task is not performed for a prolonged period of time. Similar to learning curves, forgetting curves models the relationship between the lapse in time between task repetitions and performance losses. In other words, as the lapse between task trials increases, performance degrades. Ebbinghaus (1931) was first to document forgetting curves at the individual level. Similar to learning curves, forgetting curves have been examined at the organizational level and were observed in a variety of manufacturing industries such as aircraft production (Benkard 2000), automotive assembly (Epple, Argote, & Murphy 1996), assembly of electronic appliances (Shafer, Nembhard, & Uzumeri 2001), shipbuilding (Argote, Beckman, & Epple 1990), and textile manufacturing (Nembhard 2000), as well as service industries such as franchise restaurants (Darr, Argote, & Epple 1995).

The prior literature has also documented differential forgetting curves in different organizations and industries. Argote and Epple (1990) identify employee turnover, changes in product designs or production processes, and the loss of organizational data or routines as causes of organizational forgetting that may lead to differential forgetting rates across organizations. In a similar vein, Darr et al. (1995) argued that the characteristics of individuals and organizations, the level of task specialization, employees' motivation levels, stability of employment, sophistication / complexity of product technology, and demand rates may determine how fast an organization forgets. In addition to organizational and individual characteristics, characteristics related to the task or to the type of knowledge have also been found to influence forgetting rates. Arzi and Shtub (1997) found that cognitive tasks are more prone to forgetting than mechanical (manual) labor. Schendel and Hagman (1982) found procedural skills are highly susceptible to forgetting. Similarly, in an experimental study, Bailey (1989) found that procedural tasks, which consist of discrete responses, are more subject to being forgotten than continuous control tasks, which involve repetitious movements. Nembhard (2000) showed that task complexity influences learning rates as well as forgetting rates.

Software practitioners typically deal with a variety of knowledge and skills. Each software project may require knowledge in a number of business domains, technologies, and software development methodologies. Furthermore,

[2] The categories of declarative and procedural knowledge are not necessarily pure dichotomies but can be viewed as two ends of a continuum. No knowledge in software development is purely declarative or procedural. For instance, systems development methodologies contain concepts which can be perceived as declarative (e.g., concepts of costs and quality, critical paths, risk, use cases, objects, etc.), but much of the important skills relate to how to apply various concepts and techniques in conducting and managing software projects. Similarly, technologies such as relational databases comprise of both declarative aspects (e.g., tuples, relations, and functional dependencies), which are based on sound mathematical foundations of set theory and predicate logic) as well as procedural aspects (e.g., conceptual modeling, debugging codes, and performance tuning), which are skills that are better acquired through experience.

given the transient nature of software projects, software developers must frequently transition to new projects that require knowledge in a number of new domains, technologies and methodologies. Such practices reduce the likelihood of repeating experiences with same or related knowledge, and consequently exploiting the benefits of the learning curve becomes difficult. Even if similar projects are repeated, the compound nature of projects may interfere with learning and cause forgetting (Narayanan et al. 2009).[3]

Thus, the intrinsic characteristics of software development naturally lead us to expect that forgetting curves should also exist. Since forgetting is invariant to cognitive tasks, we propose that forgetting will exist for all three types of knowledge and skills (i.e., domain, methodology, and technology knowledge).

H3a: *A software project team's performance on a current project is negatively affected by the average time interval between the current project and the prior projects in which team members experienced the same domain.*

H3b: *A software project team's performance on a current project is negatively affected by the average time interval between the current project and the prior projects in which team members experienced the same methodology.*

H3c: *A software project team's performance on a current project is negatively affected by the average time interval between the current project and the prior projects in which team members experienced the same technology.*

Would differential forgetting curves be observed for the different types of knowledge, as we have hypothesized for learning curves (i.e., Hypotheses 2)? Unfortunately, the theory of procedural vs. declarative knowledge, which was used to hypothesize about differential learning curves, does not seem to be applicable for explaining forgetting. The concept of procedural vs. declarative knowledge relates to efficacy in learning approaches – procedural knowledge is best learned via repeated trials whereas declarative knowledge through instruction or rote memorization. Although the process of forgetting simply looks like the inverse of the process of learning (i.e., un-doing of the learning), it is not a simple problem to explain the forgetting. Literature shows conflicting arguments on relationship between learning and forgetting. For example, Cochran (1968) argued that forgetting curve retrogresses along with the original learning curve. Nembhard (2000) showed that forgetting rates and learning rates are positively correlated. On the other hand, Bailey (1989) argued that forgetting rates are independent of learning rates and insisted that forgetting cannot be explained by any theory of learning. Due to lack of consistent theoretical arguments in explaining differential forgetting curves, we take an exploratory approach by investigating whether the differential rates of forgetting curves exist in software development.

Methodology

Data Collection

In order to empirically test our hypotheses, we collect and analyze an extensive archival dataset from a prominent international IT service company in Korea. The company provides contract-based custom software development and software maintenance for a variety of applications to a broad range of industries. The company has been certified by ISO 9001 and the Capability Maturity Model (CMM) since the 1990's and acquired the Capability Maturity Model Integration (CMMI) Level 5 in 2004. As a result the company has maintained detail project data since around 1995. Our dataset contains detailed information on software projects (e.g., project schedule, plan, performance, customer, industry, application type, etc.), the employees who worked on those projects, and the technologies and methodologies used in those projects.

A sample of 556 recent projects (ending between 2005 and 2007) was selected and used in the analysis. This sample of projects involved 3,341 unique employees, 6,675 employee-project assignment records, and 206,173 employee-project-technology records. The prior experiences of the employees in terms of domain, technology and methodology were computed using historical project data starting from 1988. Finally, the company's HR records were used for demographic information of employees (e.g., age, gender, education, tenure etc.)

[3] Interestingly, although learning curves have been investigated in prior studies in software development and maintenance, forgetting curves have not. To the best of our knowledge, this study is the first to investigate forgetting curves in software development.

Our sample only includes new software *development* projects. In other words, software *maintenance* projects are excluded. Although prior research on learning curves in software has primarily focused on software maintenance (e.g., Boh et al. 2007; Narayanan et al. 2009) and so investigating software maintenance would allow comparison across studies, we decided to focus on new software development projects due to several reasons. First, we believe that software development is a better context to test learning theories, especially with respect to knowledge type. Given that a dedicated team is typically assigned to maintenance tasks, there is less room for varied experiences in software maintenance. In other words, software maintainers are more likely to repeat projects in the *same* domain, *same* technology and *same* methodology than in a new software development context where there is greater variability in terms of knowledge and skills required across projects to which developers are assigned. Therefore, the software maintenance context is more conducive to learning curve effects. Consequently, observing learning curve effects in software development, where learning via cumulative experiences is more difficult, would be a stronger test of the theory. Also, given the variability of knowledge and skills across projects in software development, there is a greater likelihood that forgetting would also occur even if learning curve effects are indeed observed. Such possibilities provide a richer context in which the implications for project staffing and career paths for software professionals would be more accentuated.

Measures

Dependent variable

Several alternative measures are widely used for software development performance: on-time delivery, within-budget delivery, quality, customer satisfaction, marginal profit, etc. Data concerning customer satisfaction or system quality were not available. On-time delivery and within-budget delivery) belong to a kind of earned value measure which compares actual values with planned values. Therefore, planned values (i.e., planned project schedule and cost) hugely affect the measure. As accumulating experience, the project team is likely to make a more precise project plan. However, besides the experience, there are many other factors to affect the project plan, such as strategic relationship with customers, market competitions, and uncertainty on project, which should be controlled but were not available in our data set.

Given these difficulties, we opted to measure software project performance using actual labor costs (*LaborCost*). Each project is contracted with a fixed price, so minimizing labor cost directly increases profit for the firm. Also, because incentives are given to project members based on profit, the project team is motivated not to maintain unnecessary human resources during the project. Therefore, labor cost is an appropriate proxy for project cost performance.[4]

Independent variables

The amount of prior experiences and time lapse since prior experience for a project member in terms of domain, technology and methodology were derived from the historical project assignment data. The company defined each project with knowledge requirements in terms of domain, technology and methodology knowledge. First, *domain knowledge* for a project was categorized using 55 distinct service lines, which represent the type of application. Some service lines are vertical applications, which may exist in only one or a few industries (e.g., an e-banking system in the financial services industry) whereas others can be horizontal applications that are common across industries (e.g., HR or accounting systems). Although each project may be characterized by multiples service lines (i.e., multiple applications), the taxonomy of service lines use at the data collection site was such that most projects were defined with only one service line. Second, *technology knowledge* for a project was defined using the company's taxonomy of 5,332 distinct unit technologies (i.e., an independent technical building block) required for implementing the project. Examples of unit technology are programming languages such as Java, C, and HTML, design techniques such as Entity Relationship Diagrams (ERD) and the Unified Modeling Language (UML), and

[4] The overall project costs also include material costs for purchasing hardware and software in addition to labor cost. However, since material costs are largely defined by the characteristics of project, these are almost fixed in nature and are less affected by prior experiences. Therefore, we only use labor cost (i.e., excluding materials cost from the overall project costs) as the dependent variable.

software packages such as Oracle databases and SAP modules. The average number of unit technologies per projects was approximately 47. Finally, *methodology knowledge* was also categorized using the company's taxonomy of 64 distinct systems development methodology techniques. These included in-house developed methodologies which were proprietary to the company as well as publicly used methodologies. Examples of methodologies include object-oriented analysis and design, information engineering, component-based development, web-based development, package implementation (e.g., SAP/R3), and information strategy planning. Project management skills such as project scope, cost, time, integration, risk, and procurement were also considered methodology knowledge. Most projects used multiple unit methodologies, and the average number of unit methodology skills required per project was approximately 7.

Using the categorization scheme described above, we computed measures for amount of prior experience (*Experience*) and lapse since prior experience (*Lapse*). For each team member assigned to the focal project, we counted the number of prior projects he/she participated in that used the same domain knowledge (*ExperienceDomain*), same technology knowledge (*ExperienceTechnology*) or same methodology knowledge (*ExperienceMethodology*). We also computed the time interval (in days) between the start of the focal project and the prior project in which the same domain knowledge, same technology knowledge or same methodology knowledge was used to measure the three *Lapse* variables – *LapseDomain*, *LapseTechnology* and *LapseMethodology*. For instance, if an employee works on a project developing a retail e-banking application using a J2EE platform and the object-oriented analysis and design (OOAD) methodology, then the employee is said to have accumulated one project experience in retail e-banking (domain), J2EE platform (technology), and OOAD (methodology). Since the unit of analysis was the software development project where each project consists of several developers as part of the software project team and given the disparity in the number of distinct knowledge / skills for the three knowledge categories (domain vs. technology, vs. methodology), we used the average of individuals' experiences and lapses normalized by the number of knowledge and skills as the project team-level experience.

For example, if a software development project which requires 2 distinct methodologies (K_{M1} and K_{M2}) and 2 members (M_1 and M_2) are assigned to this project, where member M_1 was previously assigned to 3 projects which used K_{M1} or K_{M2} and member M_2 was previously assigned to 1 project which only used K_{M1}, then the total number of projects experienced by the project team is 4 and the average number per team member is 2. However, because there are 2 methodologies used in the project, the experience count is normalized by a factor of 2 (methodologies). Ultimately, the final measure for team experience for amount of methodology knowledge (*ExperienceMethodology*) becomes 1. Similarly, team experience lapse is also normalized. Suppose there are two members (M_1 and M_2) in a project team who were previously (and perhaps independently) assigned to another project that for the same domain as that of the current project. If M_1 did so 60 days ago, while the prior project of M_2 was 40 days ago, then the average interval of repeated domain experiences (*LapseDomain*) becomes 50 days.

Control variables

Project size. We used project revenue to control for project size. *ProjectSize* is computed as the total revenue minus the revenue related to materials (e.g., sever hardware, software license fees, etc.), which were excluded because they are not related with learning and workforce productivity (see footnote 4). Commonly used measures for project size include team size (i.e., number of team members), project duration, and number of function point (FP). However, since our dependent variable (*LaborCost*) is a function of team size and project duration, these two were not appropriate. Also, our data had many projects with missing FP values, so this measure could not be used due to unavailability of data.

Outsourcing / Subcontracting. Many projects in our sample subcontracted some of the activities to external outsourcing service providers. This is common practice in software development, especially when the company lacks available resources or specialized skills, as a means to hedge the risk of low utilization of human resources. Therefore, we included the ratio of labor costs for outsourcing of total labor costs (*OutsourcingRatio*) to control for the extent of such practices.

Project complexity. In order to control for the complexity of the software project, we used the number of distinct skill requirements for technology (*NumTechnologies*) and methodology (*NumMethodologies*). Given the lack of variability in number of required domain skills in our dataset, the number of domains was not included.

Macroeconomic condition. We included year dummies (for end year of project) to control for macroeconomic factors such as inflation or business cycle.

Additional control variables for organizational characteristics (e.g., sales team, development team) and team member characteristics (e.g., education level and background, gender, tenure) were initially collected but the inclusion of these variables were found not to significantly influence of enlighten our results. Therefore, we excluded these variables from our analysis and do not discuss them further. Table 2 summarizes the descriptive statistics and inter-correlations among variables.

Table 2. Descriptive Statistics and Inter-correlations

	Mean	St.dev	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
(1) *LaborCost*	19.32	1.59										
(2) *ProjectSize*	1.51	2.82	0.66***									
(3) ln(*OutsourceRatio*)	0.42	0.22	0.47***	0.16***								
(4) *NumMethodologies*	12.22	6.88	0.51***	0.49***	0.07*							
(5) *NumTechnologies*	43.97	46.80	0.60***	0.62***	0.02	0.50***						
(6) ln(*ExpDomain*)	1.09	0.84	-0.35***	-0.16***	-0.05	-0.18***	-0.27***					
(7) ln(*ExpTech*)	0.58	0.50	-0.49***	-0.30***	0.00	-0.27***	-0.43***	0.46***				
(8) ln(*ExpMethod*)	0.63	0.58	-0.45***	-0.24***	-0.05	-0.16***	-0.34***	0.43***	0.61***			
(9) *LapseDomain*	26.55	41.21	0.07	0.17***	0.04	0.07	0.01	0.13***	0.01	0.04		
(10) *LapseTech*	25.39	13.08	0.08*	0.10**	0.02	0.15***	0.01	-0.01	0.06	0.07*	0.18***	
(11) *LapseMethod*	23.53	13.61	0.11***	0.14***	-0.03	0.18***	0.13***	-0.07	-0.12***	0.19***	0.09**	0.51***

Significance Levels: * $p < 0.1$, ** $p < 0.05$, *** $p < 0.01$

Analytical Approach

The most common way to model learning curves is to use power functions. Equation 1 shows the simple power law formula which has been used to explain learning curves in the literature since it was proposed by Wright (1936). The dependent variable y is usually the cost per unit; the constant c is the initial cost per unit; variable x represents the number of trials (or cumulative experience), and finally exponent β is the learning rate parameter – β becomes negative (i.e., decreasing the cost per unit), when learning curve effects exist. Larger negative β (absolute) values imply a steeper slope of learning curves (i.e., faster learning or performance gains can be achieved with fewer cumulative experiences). Taking the natural log on both sides produces the linear form (eq. 2), which can be used with ordinary least squares (OLS) regressions.

$$y = cx^{\beta} \quad \text{(eq. 1)}$$

$$\ln(y) = \ln(c) + \beta \times \ln(x) \quad \text{(eq. 2)}$$

Consistent with prior literature on learning curves in software development (Boh et al. 2007; Langer et al. 2008; Huckman et al. 2009), we develop the following model for project i:

Model 1: $\ln(LaborCost_i) = \beta_0 + \beta_1 \times ProjectSize_i + \beta_2 \times \ln(OutsourcingRatio_i) + \beta_3 \times NumMethodologies_i + \beta_4 \times NumTechnologies_i + \beta_5 \times Y2005_i + \beta_6 \times Y2006_i + \beta_7 \times \ln(ExperienceDomain_i) + \beta_8 \times \ln(ExperienceTechnology_i) + \beta_9 \times \ln(ExperienceMethodology_i) + \varepsilon_i$

In order to incorporate forgetting effects, we add the experience lapse variables (eq. 3) and take the natural log on both sides:

$$y = c(x\alpha^t)^{\beta} \quad \text{(eq. 3)}$$

$$\ln(y) = \ln(c) + \beta \times \ln(x) + \beta \times \ln(\alpha) \times t \quad \text{(eq. 4)}$$

The variables y, x, c and β are the same as in (eq. 1). Variable t represents the time elapsed since previous trial and α is the forgetting rate parameter. When forgetting curve effects exist, α takes a value between 0 and 1. $\alpha = 1$ means no forgetting while $\alpha = 0$ means no accumulation. Smaller values of α imply a steeper slope of forgetting curves (i.e., faster forgetting). Model 2 below operationalizes the knowledge depreciation model where the amount of forgetting is related to both the amount of accumulated experience and elapsed time (Argote et al. 1990; Boone, Ganeshan, & Hicks 2008; Darr et al. 1995).

Model 2: $\ln(LaborCost_i) = \beta_0 + \beta_1 \times ProjectSize_i + \beta_2 \times \ln(OutsourcingRatio_i) + \beta_3 \times NumMethodologies_i + \beta_4 \times NumTechnologies_i + \beta_5 \times Y2005_i + \beta_6 \times Y2006_i + \beta_7 \times \ln(ExperienceDomain_i) + \beta_8 \times \ln(ExperienceTechnology_i) + \beta_9 \times \ln(ExperienceMethodology_i) + \beta_7 \times \ln(\alpha_{10}) \times LapseDomain_i + \beta_8 \times \ln(\alpha_{11}) \times LapseTechnology_i + \beta_9 \times \ln(\alpha_{12}) \times LapseMethodology_i + \varepsilon_i$

Econometric Issues

To deal with potential heteroskedasticity problems, we compute White's heteroskedasticity-consistent estimates of the regression model parameters, correcting for inefficiencies due to unequal error variances (White 1980). Finally, we test for potential multicollinearity problems by checking that variance inflation factors (VIF) for the right-hand side variables (i.e., independent and control variables) are within recommended limits.

Analysis and Results

We conducted the analyses using a hierarchical approach. We first estimate a baseline regression model that only includes the control variables (Baseline Model). Then we progressively add the independent variables for cumulative experience (*ExperienceDomain*, *ExperienceTechnology*, and *ExperienceMethodology;* Model 1) and experience lapse (*LapseDomain*, *LapseTechnology*, and *LapseMethodology*; Models 2) and check to see if the inclusion of the independent variables increases the explanatory power of the regression models. The results are summarized in Table 3. The increased explanatory power between models shows that both cumulative experience and experience lapse variables are predictors of the project performance gains, labor cost (Base model vs. Model 1: $\Delta R^2 = 0.083$, Model 1 vs. Model 2: $\Delta R^2 = 0.027$).

Table 3. Regression Results

	Baseline Model		**Model 1**		**Model 2**	
	Parameter Est.	Std Err	Parameter Est.	Std Err	Parameter Est.	Std Err
Intercept	17.109***	0.1268	18.123***	0.1449	18.160***	0.1641
ProjectSize	0.178***	0.0354	0.167***	0.0293	0.148***	0.0169
ln(*OutsourcingRatio*)	2.891***	0.2096	2.879***	0.1825	2.851***	0.1744
NumMethodologies	0.036***	0.0068	0.033***	0.0057	0.031***	0.0065
NumTechnologies	0.011***	0.0017	0.007***	0.0013	0.006***	0.0011
Y2005	-0.221**	0.1010	-0.393***	0.0887	-0.407***	0.0978
Y2006	-0.187*	0.0987	-0.237***	0.0863	-0.214***	0.0917
ln(*ExpDomain*)			-0.188***	0.0482	-0.236***	0.0586
ln(*ExpTechnology*)			-0.450***	0.1007	-0.494***	0.1113
ln(*ExpMethodology*)			-0.389***	0.0770	-0.436***	0.0982
LapseDomain					0.999	0.0038
LapseTechnology					0.998	0.0068
LapseMethodology					0.981***	0.0076
Sample Size (*N*)	556		556		454	
R^2	0.6665		0.7492		0.7769	
Adj R^2	0.6641		0.7451		N/A	

Significance Levels: * $p < 0.1$, ** $p < 0.05$, *** $p < 0.01$
Notes: Robust standard errors are reported. R^2 for Model 2 is pseudo-R^2.

In the baseline model, we first find that our control variables are significantly associated with performance gains. Larger projects (*ProjectSize*) and more complex projects (*NumMethodologies* and *NumTechnologies*) are associated with increased labor cost (*ProjectSize*: $\beta_1 = 0.178$, $p < 0.01$; *NumMethodologies*: $\beta_3 = 0.036$, $p < 0.01$; *NumTechnologies*: $\beta_4 = 0.11$, $p < 0.01$). *OutsourcingRatio* is positively correlated with labor cost ($\beta_2 = 2.879$, $p < 0.01$), and both year dummies are representing the inflation effect (*Y2005*: $\beta_5 = -0.221$, $p < 0.01$; *Y2006*: $\beta_6 = -0.187$, $p < 0.1$).

With Model 1, we first observe that the coefficients of all control variables remain relatively stable – another indicator that multicollinearity is not at play. More substantively, we find that all three of the cumulative experience

variables have significant effects on project performance gains (*ExperienceDomain*: β_7 = -0.188, $p < 0.01$; *ExperienceTechnology*: β_8 = -0.450, $p < 0.01$; and *ExperienceMethodology*: β_9 = -0.389, $p < 0.01$). In other words, all three categories of knowledge and skills for software development exhibit learning curve effects. Hypothesis 1 was thus supported.

Using the Wald test, we further checked whether the different types of software development knowledge (i.e., domain vs. technology vs. methodology) have differential learning curves. The coefficient for *ExperienceMethodology* was found to be significantly less than that of *ExperienceDomain* (χ^2 = 4.17, $p < 0.05$). Similarly, the coefficient for *ExperienceTechnology* was significantly less than that of *ExperienceDomain* (χ^2 = 4.64, $p < 0.05$). However, the difference between *ExperienceMethodology* and *ExperienceTechnology* was not significant (χ^2 = 0.16, ns). These results imply that methodology knowledge and technology knowledge have steeper learning curves than domain knowledge but there was no difference in the slopes between the learning curves for methodology and technology knowledge. Hypothesis 2 was partially supported – H2a and H2b were supported, but H2c was not supported.

Nonlinear regression techniques[5] were used to estimate Model 2. We observe again that the coefficients of all control variables, as well as cumulative experience variables remain relatively stable. With respect to the experience lapse variables, we find that longer lapses between experiences with the same methodology have at least a marginal effect on project performance degradation (*LapseMethodology*: α = 0.981, $p < 0.01$). However, lapses in domain experience or technology experience did not seem to degrade project performance (*LapseDomain*: α = 0.999, ns; *LapseTechnology*: α = 0.998, ns). This suggests that knowledge and skills on methodology acquired through cumulative experiences may be prone to forgetting, while knowledge of domain and technology would be more resilient to forgetting. In summary, Hypothesis 3 is only partially supported – H3c was supported, while H3a and H3b were not supported.

Robustness of Results

The robustness of these results was tested using a variety of additional analyses. First, to test whether the results were sensitive to the inclusion/exclusion of some control variables, we ran the analysis progressively by including different combinations of control variables. The results of the key independent variables – the experience and lapse variables – were found to be stable and were not affected by adding or dropping control variables. In addition, the Wald tests results for differential learning curves (for testing Hypotheses 2) was also largely consistent with the results reported herein.

To test whether forgetting effects was robust to model specification, we tested the forgetting model using the power function of time, which is another widely used function for fitting forgetting curves (Wixted and Ebbesen 1991). The results were similar to our original results with the exception that forgetting curves were also found for technology (in addition to methodology). This result provides further confidence that forgetting effects do exist in software development, especially for methodology knowledge and also perhaps for technology knowledge.

Finally, we checked whether the results were robust to operationalization of the experience variables. In our original analysis, we have used the *counts* of prior experiences in the same domain (or same technology or same methodology) to measure prior experiences. However, this measure does not take into account the difference between shorter and longer projects. For example, with our original having completed 2 projects that lasted 6 months each is deemed to provide more experience than having completed 1 project that lasted 1 year. To check whether this may be a concern, we developed an alternative measure for cumulative experience for a particular knowledge and skill by computing the cumulative duration (i.e., in working days) for which an individual has worked on the particular knowledge and skill. Using this measure yielded similar results.

Given the results of these additional analyses, we conclude that our results are quite robust.

[5] Proc NLIN in SAS 9.1.3 was used to estimate Model 2. Using general optimization approaches (e.g., the Gauss-Newton method), this procedure iteratively searches for the parameter values that produces the lowest residual sum of squares.

Conclusion and Discussion

This study examined learning and forgetting curves in software development. We ask whether different types of knowledge and skills used in software development, namely domain knowledge, technology knowledge, and methodology knowledge, benefit from learning via cumulative experiences and/or suffer from forgetting by lapse of time, and if so, whether there are any differences in their respective learning and forgetting rates. Our results suggest that all three types of software development knowledge exhibit learning curve effects but methodology and technology knowledge are more efficiently learned via cumulative experiences than domain knowledge and that only methodology knowledge exhibits a forgetting curve while domain and technology knowledge do not.

Our paper makes several contributions to literature. First, we include forgetting effects in our model and analyses. Given that software developers typically engage in unrelated projects in between related projects, it is important to integrate learning and forgetting in the same model. To the best of our knowledge, our study is the first to investigate forgetting effects in software development. Second, we showed that the type of knowledge can also entail differential learning curves. Although extensive research has investigated differential learning curves, few have actually looked at the impact of the types of knowledge.

Our empirical findings also shed light into practical questions in software development such as *which knowledge and skills are most influential to project performance*, *how to staff software projects*, and *how to think about career development paths for software professionals*. For example, software organizations may infer that methodology and technology experience is more influential to performance than domain experience or that developing a domain expert takes relatively longer than developing a technology or methodology guru due to different learning rates.

This paper is however not without limitations. One of limitations is the use of labor cost as the dependent variable representing software project performance. Although there are several different measures used for software development performance such as quality, time, and customer satisfaction, our study only includes cost performance due to the limited availability of data and implications of the strategic context of the company. Another limitation lies in our conceptualization and modeling of the mechanism of forgetting. Forgetting is caused not only by lapses in time between trials but also from interference by other experiences between successive similar trials. In this paper, we only consider the former. The impact of interference should also be investigated in future research.

In addition to the obvious future research directions that stem from the aforementioned limitations, this paper opens up many new avenues for future research. Having shown that the type of knowledge matters in learning and forgetting, it would be interesting to investigate whether breadth or diversity/variety of prior experiences have an impact on project team performance. We may also consider the composition of the project team (in terms of knowledge) and investigate its impact on project performance. For example, are teams composed of mainly specialists better than teams composed of many generalists? What is an appropriate balance of generalists and specialists in the software team? This paper is but a small step in this exciting new direction.

References

Abraham, T., Beath, C., Bullen, C., Gallagher, K., Goles, T., Kaiser, K., and Simon, J. (2006) "IT workforce trends: Implications for IS programs," *Communications of the AIS* (17:1) 1147-1170.

Anderson, J.R. (1982) "Acquisition of cognitive skill," *Psychological Review* (89:4) 369-406.

Anderson, J.R. (1983) *The architecture of cognition*, Harvard University Press, Boston: MA.

Argote, L., Beckman, S.L., and Epple, D. (1990) "The persistence and transfer of learning in industrial settings," *Management Science* (36:2) 140-154.

Argote, L. and Epple, D. (1990) "Learning-curves in manufacturing," *Science* (247:4945) 920-924.

Armstrong, D.J. and Hardgrave, B.C. (2007) "Understanding mindshift learning: The transition to object-oriented development," *MIS Quarterly* (31:3) 453-474.

Arzi, Y. and Shtub, A. (1997) "Learning and forgetting in mental and mechanical tasks: A comparative study," *IIE Transactions* (29:9) 759-768.

Bailey, C.D. (1989) "Forgetting and the learning curve: A laboratory study," *Management Science* (35:3) 340-352.

Bassellier, G. and Benbasat, I. (2004) "Business competence of information technology professionals: Conceptual development and influence on it-business partnerships," *MIS Quarterly* (28:4) 673-694.

Benkard, C.L. (2000) "Learning and forgetting: The dynamics of aircraft production," *American Economic Review* (90:4) 1034-1054.

Boh, W.F., Slaughter, S.A., and Espinosa, J.A. (2007) "Learning from experience in software development: A multilevel analysis," *Management Science* (53:8) 1315-1331.

Boone, T., Ganeshan, R., and Hicks, R.L. (2008) "Learning and knowledge depreciation in professional services," *Management Science* (54:7) 1231-1236.

Brooks, F.P. (1987) "No silver bullet: Essence and accidents of software engineering," *IEEE Computer* (20:4) 10-19.

Chan, C.L., Jiang, J.J., and Klein, G. (2008) "Team task skills as a facilitator for application and development skills," *IEEE Transactions on Engineering Management* (55:3) 434-441.

Cochran, E.B. (1968) *Planning Production Costs: Using the Improvement Curve*. Chandler Publishing, San Francisco: CA.

Darr, E.D., Argote, L., and Epple, D. (1995) "The acquisition, transfer, and depreciation of knowledge in service organizations: Productivity in franchises," *Management Science* (41:11) 1750-1762.

Dutton, J.M. and Thomas, A. (1984) "Treating progress functions as a managerial opportunity," *Academy of Management Review* (9:2) 235-247.

Ebbinghaus, H. (1913) *Memory: A contribution to experimental psychology*, Teachers College, Columbia University, New York: NY.

Epple, D., Argote, L., and Murphy, K. (1996) "An empirical investigation of the microstructure of knowledge acquisition and transfer through learning by doing," *Operations Research* (44:1) 77-86.

Faraj, S. and Sproull, L. (2000) "Coordinating expertise in software development teams," *Management Science* (46:12) 1554-1568.

Gagne, E.D., Yekovich, C.W., and Yekovich, F.R. (1985) *The cognitive psychology of school learning*, Little Brown and Co., Boston: MA.

Goles, T., Hawk, S., and Kaiser, K.M. (2008) "Information technology workforce skills: The software and IT services provider perspective," *Information Systems Frontiers* (10:2) 179-194.

Greene, W. H. (2008) *Econometric Analysis*, 6th ed., Prentice-Hall, Upper Saddle River: NJ.

Herschbach, D.R. (1995) "Technology as knowledge: implications for instruction," *Journal of Technology Education* (7:1) 31-42.

Huckman, R.S., Staats, B.R., and Upton, D.M. (2009) "Team familiarity, role experience, and performance: evidence from Indian software services," *Management Science* (55:1) 85-100.

Jones, B.F. and Idol, L. (1990) *Dimensions of thinking and cognitive instruction*, Lawrence Erlbaum Associates, Hillsdale: NJ.

Langer, N., Slaughter, S.A., and Mukhopadhyay, T. (2008) "Project managers' skills and project success in IT outsourcing," in *Proceedings of the Twenty Ninth International Conference on Information Systems*, Paris, France.

Lee, D.M.S., Trauth, E.M., and Farwell, D. (1995) "Critical skills and knowledge requirements of IS professionals: A joint academic-industry investigation," *MIS Quarterly* (19:3) 313-340.

Martin, J. (1991) *Information engineering*, Prentice Hall, Englewood Cliffs: NJ.

McCormick, R. (1997) "Conceptual and procedural knowledge," *International Journal of Technology and Design Education* (7:1) 141-159.

Narayanan, S., Balasubramanian, S., and Swaminathan, J. (2009) "A matter of balance: Specialization, task variety, and individual learning in a software maintenance environment," *Management Science*, forthcoming.

Nembhard, D.A. (2000) "The effects of task complexity and experience on learning and forgetting: A field study," *Human Factors* (42:2) 272-286.

Nonaka, I. (1991) "The knowledge creating company," *Harvard Business Review* (69:6) 96-104.

Pisano, G.P., Bohmer, R.M.J., and Edmondson, A.C. (2001) "Organizational differences in rates of learning: Evidence from the adoption of minimally invasive cardiac surgery," *Management Science* (47:6) 752-768.

Polanyi, M. (1967) *The Tacit Dimension*, University of Chicago Press, Chicago: IL.

Reagans, R., Argote, L., and Brooks, D. (2005) "Individual experience and experience working together: Predicting learning rates from knowing who knows what and knowing how to work together," *Management Science* (51:6) 869-881.

Schendel, J.D. and Hagman, J.D. (1982) "On sustaining procedural skills over a prolonged retention interval," *Journal of Applied Psychology* (67:5) 605-6 10.

Schilling, M.A., Vidal, P., Ployhart, R.E., and Marangoni, A. (2003) "Learning by doing something else: Variation, relatedness, and the learning curve," *Management Science* (49:1) 39-56.

Shafer, S.M., Nembhard, D.A., and Uzumeri, M.V. (2001) "The effects of worker learning, forgetting, and heterogeneity on assembly line productivity," *Management Science* (47:12) 1639-1653.

Simon, J.C., Kaiser, K.M., Beath, C., Goles, T., and Gallagher, K. (2007) "Information technology workforce skills: Does size matter?" *Information Systems Management* (24) 345-358.

Vincenti, W.G. (1984) "Technological knowledge without science: The innovation of flush riveting in American airlines, C1930-C1950," *Technology and Culture* (25:3) 540-576.

Wixted, J.T. and Ebbesen E.B. (1991) "On the form of forgetting," *Psychological Science* (2:6) 409-415.

White, H. (1980) "A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity," *Econometrica* 48(4) 817-838.

Wiersma, E. (2007) "Conditions that shape the learning curve: Factors that increase the ability and opportunity to learn," *Management Science* (53:12) 1903-1915.

Wright, T.P. (1936) "Factors affecting the cost of airplanes," *Journal of the Aeronautical Sciences* (3) 122-128.

Yelle, L. (1979) "The learning curve: Historical review and comprehensive survey," *Decision Sciences* (10:2) 302-328.